Three-dimension neural network chip with Staking Memories method

Dang Nam Khanh, Ben Abdallah Abderazek, Nguyen Ngo Doanh

Version 1.0, Oct 20, 2022

BACKGROUND

1. This invention is about a three-dimensional spiking neural network inference chip, and more specifically, to memory stacking method with dynamic quantization for spiking neural networks.

SUMMARY

1. According to the present invention, spiking neural inference processors are provided. A spiking neural network inference processor comprises a plurality of Neural Computing Cores (205),andone Network-on-Chip (204) interconnecting the plurality of Neural Computing Cores.
2. Each of the plurality of Neural Computing Cores (205) comprises a Network Interface (203) to interconnect with the Network-on-Chip (204), a Weight Memory (206) to store parameters and weight, and a Neuron Array (212) to perform the operation of the neuron model.
3. The conventional hardware neuromorphic chip (101) usually divides the neurons into multiple clusters of neurons (Neural Computing Cores: 102). Each cluster usually consists of the same number of neurons. Neuron clusters communicate via an interconnect infrastructure. Weights (strength of neuron connections) are stored in memory (105). The neuron circuit and its weights of the incoming connections are considered as the neuron model.
4. In this invention, the neuromorphic inference chip (213) is realized in several silicon layers. The first layer (Computing Layer 201) is dedicated to computing and communication modules and the upper layers (Memory Layer 207) are for memories. Inter-layer vias (209) is used to communicate between layers.
5. In this invention, the value of weight is not stored in a single memory. Each value of weight is stored in a plural number of Memory Weights (206). In general, the weight of a connection (303) in a neural network (301) is quantized into a fixed-point format (305). This is the quantization process (304). The quantized weight (305) is decomposed into a plural number of Partial Quantized Weights (307). This is the decomposition process (306). Each Partial Quantized Weight (307) is stored in separated Memory Weight of different Memory Layer (207).
6. In this invention, the Neuron Array (212) reads the weight from upper Memory Layers (207) via Inter-layer Vias.
7. This invention provides a method for low-power inference:
   1. If there is not enough power supply, the voltage of the upper layer is reduced.
   2. If the voltage of the upper layer cannot be reduced anymore, the upper layer will be turned off. The weights are further quantized to smaller bit format. The neuromorphic inference chip (213) still performs its inference with possibly lower accuracy.
   3. If there is enough power supply, memory at the upper layers will be restored. The inference chip will use more bits in weights for computation.
8. This invention provides fault tolerance in weight memories.
   1. If there are errors in one of the weight memories. The failed layer will be removed from the reading process. The weights are only stored in healthy memories and the inference chip performs with a smaller bit format.

BRIEF DESCRIPTION OF THE DRAWINGS

1. FIG 1 is an exemplary of a conventional neuromorphic inference chip.
2. FIG 2 is an exemplary of the proposed neuromorphic inference chip.
3. FIG 3 is the quantization process.
4. FIG 4 is an example of the dynamic quantization process and impact of voltage scaling.
5. FIG 5 is the mapping location of the partial quantized weight (307) which is the result of the dynamic quantization.
6. FIG 6 is the mode selection flow chart.
7. FIG 7 is the low-power mode flowchart.
8. FIG 8 is the high-performance mode flowchart.
9. FIG 9 is the conventional adder.
10. FIG 10 is our proposed adder with dynamic quantization.
11. FIG 11 is the potential application.
12. FIG 12 is the method of reallocating partial weight to tolerate faults.
13. FIG 13 is the fault-tolerant flowchart.

DETAILED DESCRIPTION OF EMBODIMENTS

[PART-I: DESCRIPTION OF THE PROPOSED SYSTEM]

[BACK GROUND]

1. FIG.1 illustrates an exemplary conventional (prior-art) neuromorphic inference chip. The neuromorphic inference chip (101) consists of two main parts: Neural Computing Core (102)and NoC (Network-on-Chip) communication (103). For management the neuromorphic chip, it is connected with host PC (109)via an inter-chip interconnect (108).
2. The Neural Computing Core (102) consists of three main parts: Network Interface (104), Neuron Array (106), and Weight Memory (105). Network Interface (104) is connected to the NoC router (107) and handles the communication between the Neural Computing Core (102) and the Network-on-Chip (103). Network-on-Chip (103) is a set of NoC router (107) connected in a certain topology.
3. Inputs from the Network-on-Chip (103) are delivered by the NoC router (107) to the Network Interface (104). The Network Interface (104) decodes them for the computation. The decoded inputs are fed into Weight Memory (105) and Neuron Array (106) to perform the computation of neurons. Outputs of the computation are stored, encoded, and sent back to the NoC router (107) in order to deliver to the destination Neural Computing Core. A set of NoC routers is structured in a topology (for example: 2D Mesh) to allow communication between Neural Computing Core.
4. A Neuron Array (106) consists of a plural number ofPhysical Neuron (SN) (110). Each Physical Neuron is indexed by a number (SN1, SN2, … SNN) (N: number of SNs).
5. Each Physical Neuron (110) can receive different inputs from the Network Interface (104) and different weights from Weight Memory (105).
6. In each time step, a Physical Neuron (110) takes input from the origin (sending Physical Neuron (110)) through its incoming Weight Memory Block of Physical Neuron (111). Here we call them (M: number of incoming neuron. We also call the weight of each edges . Neuron Computing Model has an activation function called . Output of Physical Neuron (110) is as follows:

[CONVENTIONAL MEDTHOD]

1. Conventional systems store memory weight (105) in the same layer with computation – Neuron Array (106).
2. Conventional systems when operating at low-power mode rely on several techniques: (1) clock gating the circuit, and (2) power gating the circuit.
3. Patent [10] uses distributed storage multiple cores systems connected via an on-chip network. Each core stores weights for computation with the input fed from the on-chip network. Our system also uses distributed weights (no reload weight during operation).
4. Patent [11] illustrates a parallel spiking neural network using an on-chip network. Each core utilizes a plurality of adders and multipliers to accelerate the computation.
5. Patent [12] extends the scalability of multi-cores Network-on-chip based with chip-to-chip connection.
6. Patent [13] is the method to route spikes inside the Spiking Neural Network using Network-on-Chip.

[LIMITATION OF CONVENTIONAL METHODS]

1. Since the memory weight (105) takes the major part of the neuromorphic chip (101), the dimension of the memory weight (105) leads to a long distance between the NoC routers (107).
2. However, it is not possible to partially power-gating the memory weight. Once the memory weight is turned off, the whole memory is not accessible.

[PROPOSED METHOD]

1. FIG. 2 illustrates an exemplary chip of realization of the patent. The neuromorphic chip (213) consists of a plural number of silicon layers. One of the layers is **the** Computing Layer (201) which has all the computation circuits and I/O with other devices, chips, or a host PC. The other layers are Memory Layers (207) which have all the weight memories. The layers are connected via Inter-layer Vias (209) connection. The operations of the Spiking Neural Network Model (11) can be expressed as parallel operation of individual Neuron Computing Model (12). Individual Neuron Computing Model (12) operates synchronously by following an interval called time-step.
2. The memory weights (206) are placed in other layers; therefore, the computing layer (201) consists of only neuron array (212), NoC router (202) and Network-Interface (203) which can have smaller foot print. Distance between NoC routers (202) will be reduced.
3. Memory weights are stored in different layers. Each weight is partitioned into a collection of sub-set bits. Each sub-set bit is stored in a different layer.
4. To reduce the power consumption, the least significant sub-set bits can be first undervolted (reducing its supply voltage) or completely removed by turning off the corresponding layers. Both the static and dynamic power of the layer will be reduced.

**[Claim]**

1. This document presents a proposed system with stacking Memory Layers apart from a separated Computing Layer. To adapt to different power scenarios, this patent provides a Dynamic Quantization solution.
2. To reduce the power consumption of an AI inference, reducing the voltage of the layer can be applied. If more power reduction is needed, reducing the number of bits in weights can be done by turning off one of the Memory Layer. This also reduces the performance (accuracy) of the AI inference as a trade-off to the low power consumption.
3. To resume the high-power high-performance, restoring the turning off Memory layer is needed.

[PART-II: DESCRIPTION OF DYNAMIC QUANTIZATION AND EFFECT OF VOLTAGE SCALING]

1. This part describes the provided **Dynamic Quantization** method.
2. FIG. 3 illustrates the quantization process from the neural network model into fixed point format. The method consists of two phases: quantization process (304) and decomposition (306).
3. In the conventional quantization process (304) a float weight (303) is converted into a quantized weight (305) with a fixed number of bit (i.e., 8-bit in 305).
4. In this proposal, a decomposition process (306) divides the fixed number of bits into subsets. For example, FIG.3 illustrates four subsets of 2 bits (307) from a quantized weight (305).
5. FIG.4 illustrates different scenarios of the dynamic quantization process.
6. 401 shows the 8-bit quantization and 2-subset decomposition. Each subset is 4-bit.
7. 402 shows the effect of voltage reduction where bits can be flipped (0 to 1 or 1 to 0). 403 shows the value of flipped bit in decimal.
8. The dynamic quantization follows the rules below:
9. A float weight (303) is quantized into **Q**-bit weight in fixed point format (305).
10. A Q-bit weight (305) is decomposed into N subsets of -bit (i = 1,2,3… N):
11. A Q-bit weight (305) can be obtained by merging the N subsets.
12. During inference, each sub-set is stored in a different Memory Layer (207).
13. FIG 5. illustrates that the quantized weight (305) is decomposed into two partial quantized weights (307). First partial quantized weight is stored in the second layer close to the computing layer. The second partial quantized weight is stored in the third layer.

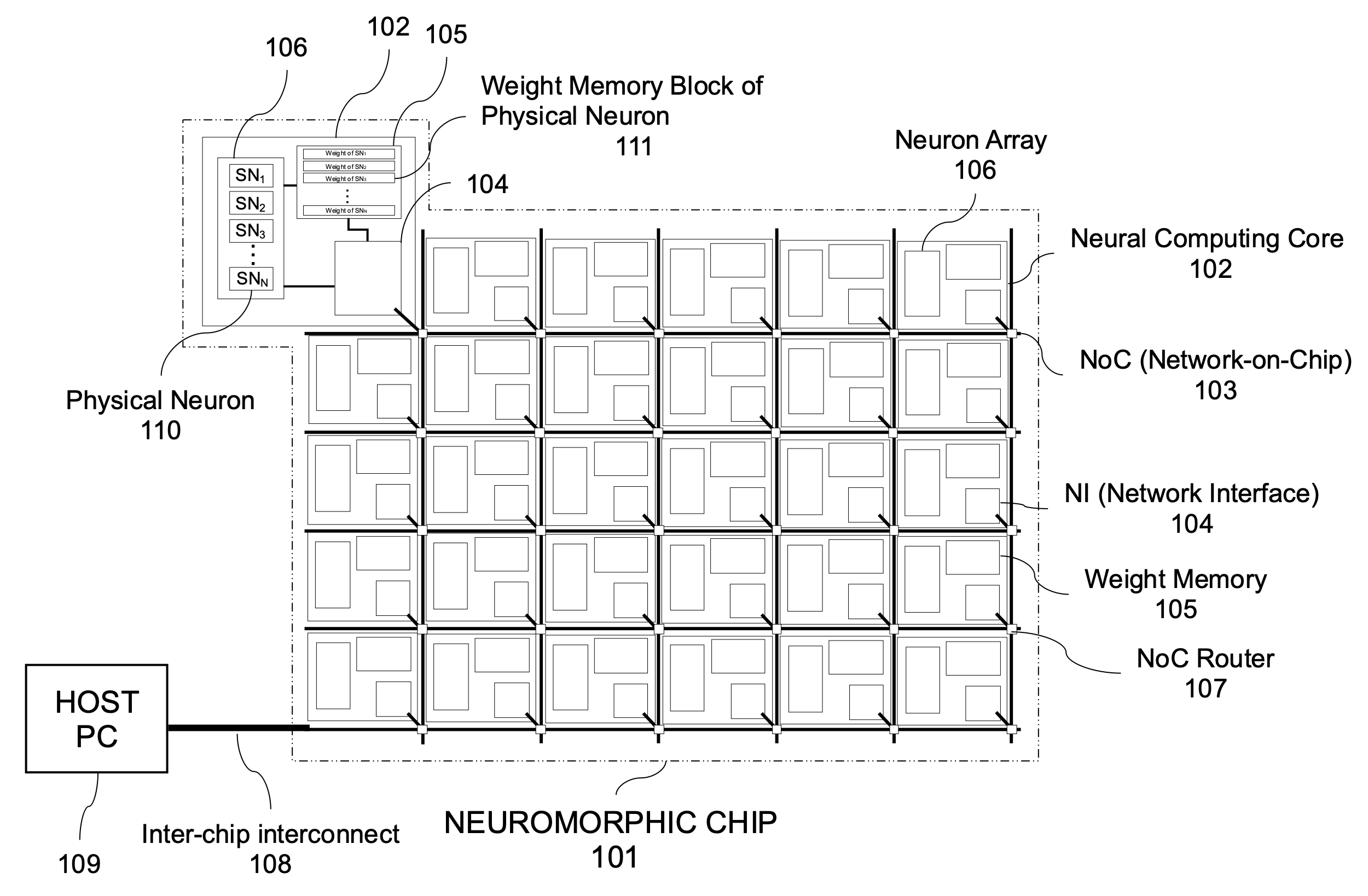
[PART-II: DESCRIPTION OF CONTROLLING PROCESS]

1. FIG 6. illustrates the flow chart of dynamic quantization to adapt to different power consumption. After it starts, it first compares the supplied power () and the consumed power (). If there is more supplied power, the system can enter high-performance mode (604). If there is less supplied power, the system enters low-power mode (605). Otherwise, the system stays unchanged (603).
2. The system will wait for a time **T** (606)before starting to check the power comparison again (600).
3. After entering low-power mode (604), the flow chart in FIG. 7 is performed. It starts with the highest active memory layer **k** (active layer with the least significant bits) (701). It compares the supplied and consumed power (702), and if it lacks power, it starts to reduce the power consumption. If , it finishes.
4. First, 708 checks if the voltage of layer **k** () is equal to the lower limitation of scaled voltage (). If , it reduces the voltage of layer **k** (709), and back to the power comparison (702).
5. If voltage scaling is not possible (). The system checks if the active layer is the last memory layer (k=1). If this is the last layer, it is not possible to turn off (706) and the algorithm finishes (707).
6. If k is not the last layer, the system turns off the layer k completely (703). Then, the index k is reduced by 1 (704), and the system is back to power comparison (702).
7. After entering high-performance mode (606), the flow chart in FIG. 8 is performed. It starts with the highest active memory layer **k** (active layer with the least significant bits) (701). N is the number of memory layers. The flow chart first compares the power supply and power consumption (802).
8. If there is a power budget, it starts to check if the voltage of layer **k** () is maximized (811). If not, it increases the voltage of layer **k** (810) and compare the power again (802).
9. If the voltage is maximized, it checks if the active layer is the highest layer (803). If the condition is true, there is no layer to be turned on and the algorithm finishes (809).
10. If **k** is not the highest layer, the value of **k** is increased (804). and it is turned on the layer **k** (805).
11. FIG. 9 illustrates a conventional adder. In general, the result (out) is generated right after having two input values (in1 and in2).
12. FIG 10 illustrates the modified adder to support turning off layers. Here, multiplexers (1008) are added to allow inserting zeros instead of values from wires. In normal conditions, the value in1 is the concatenated value of pw2 and pw1. If layer 2 is turned off, it means the value of pw2 is not accessible, and on2 switches to “0000”.
13. As shown in the waveform (1007), when the value of pw2 becomes high-impedance **(Z)**, the value 0000 is selected instead.

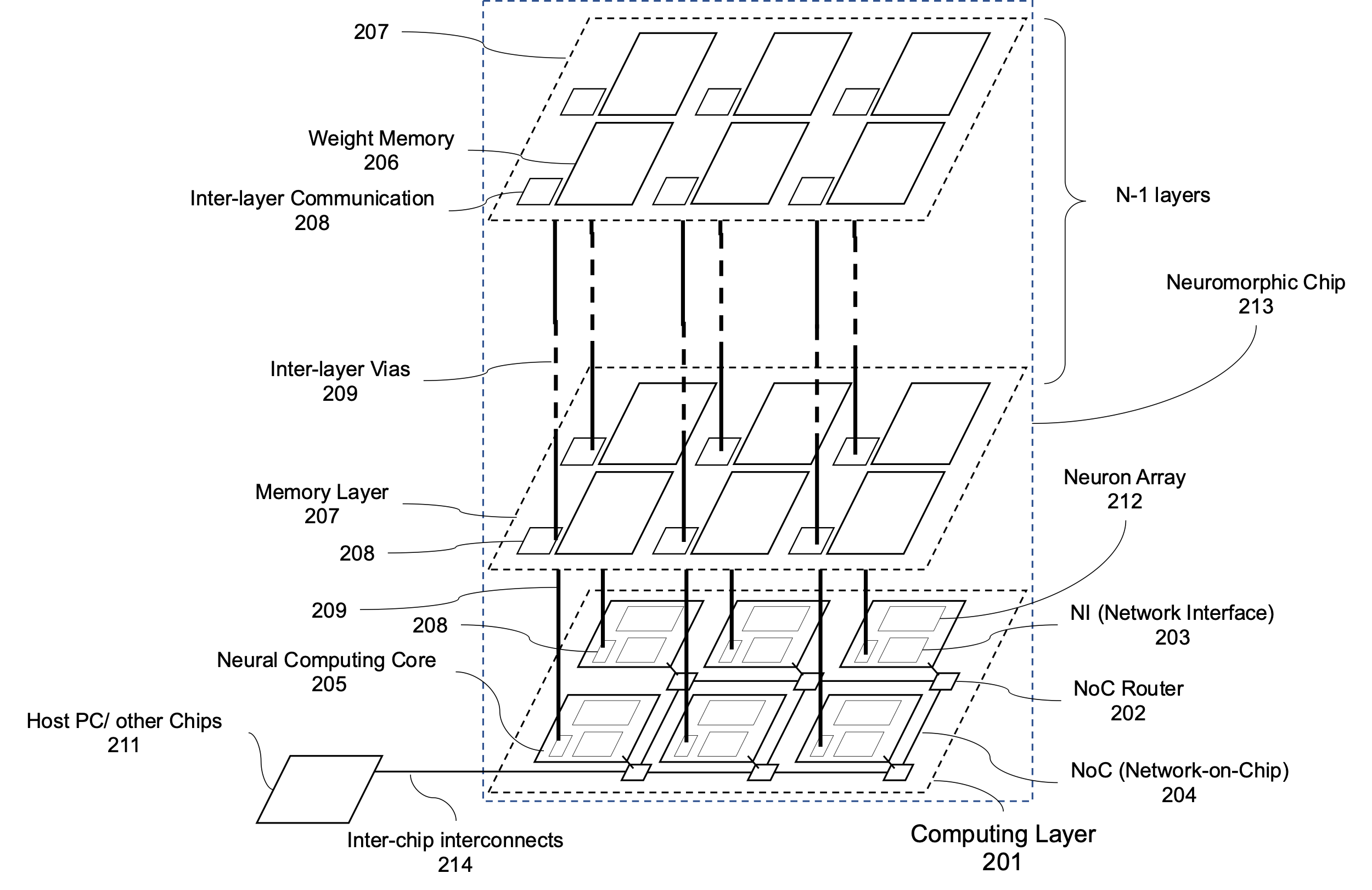
[PART-II: DESCRIPTION OF MEMORY-ERROR TOLERANCE CONTROLLING]

1. FIG 11 illustrates an explementary of the usage of memory-error tolerance controlling for prosthetic hands.
2. Medical application, such as prosthetic hand, has high vulnerability with permanent defects in its controlling devices.
3. A defective memory cell at more significant bits can affect the accuracy of the neuromorphic chip.
4. FIG 12 illustrates the fault-tolerant method. Once a partial faulty weight becomes faulty (1200), the system reallocates into another layer and removes the faulty weight cell from the computation. The partial weight is reallocated in 1201.
5. FIG 13 is the flowchart for reallocating faulty partial weight. It first runs a memory check (1301). Then, it checks whether no faulty memory cell (1302), if there is no faulty memory, it finishes. If there is a faulty memory cell, it lists all the faulty weights (1303) as a set. F is the number of faults.
6. The flowchart starts from index (1304). Then, the flowchart checks if the fault is in the LSB (least significant bit) layer. If the condition is yes, no correction is provided, and the flowchart finishes (1309).
7. If the fault is NOT in the LSB, it reallocates from the faulty layer to LSB (1306) as shown in FIG 12. Then, the index is increased (1307) until the last one (1308). After the last fault is checked, the algorithm finishes (1309).

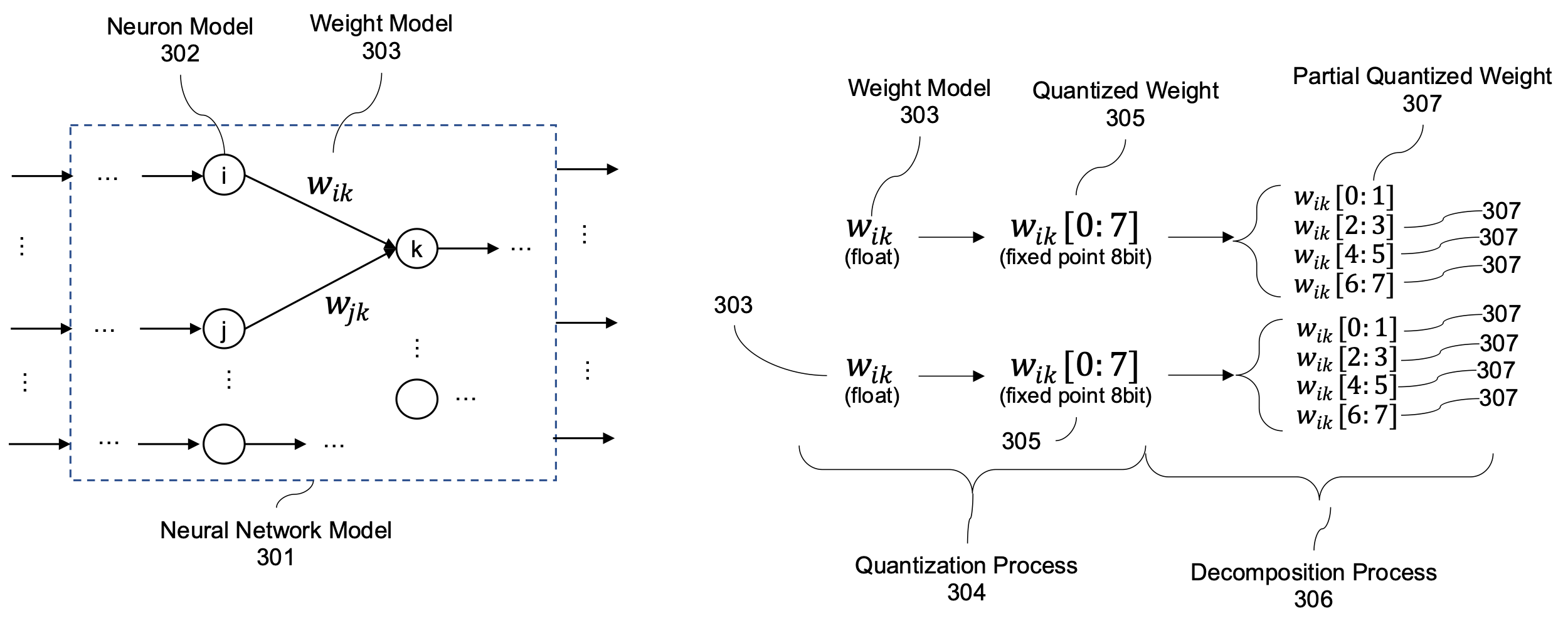
**FIGURES**



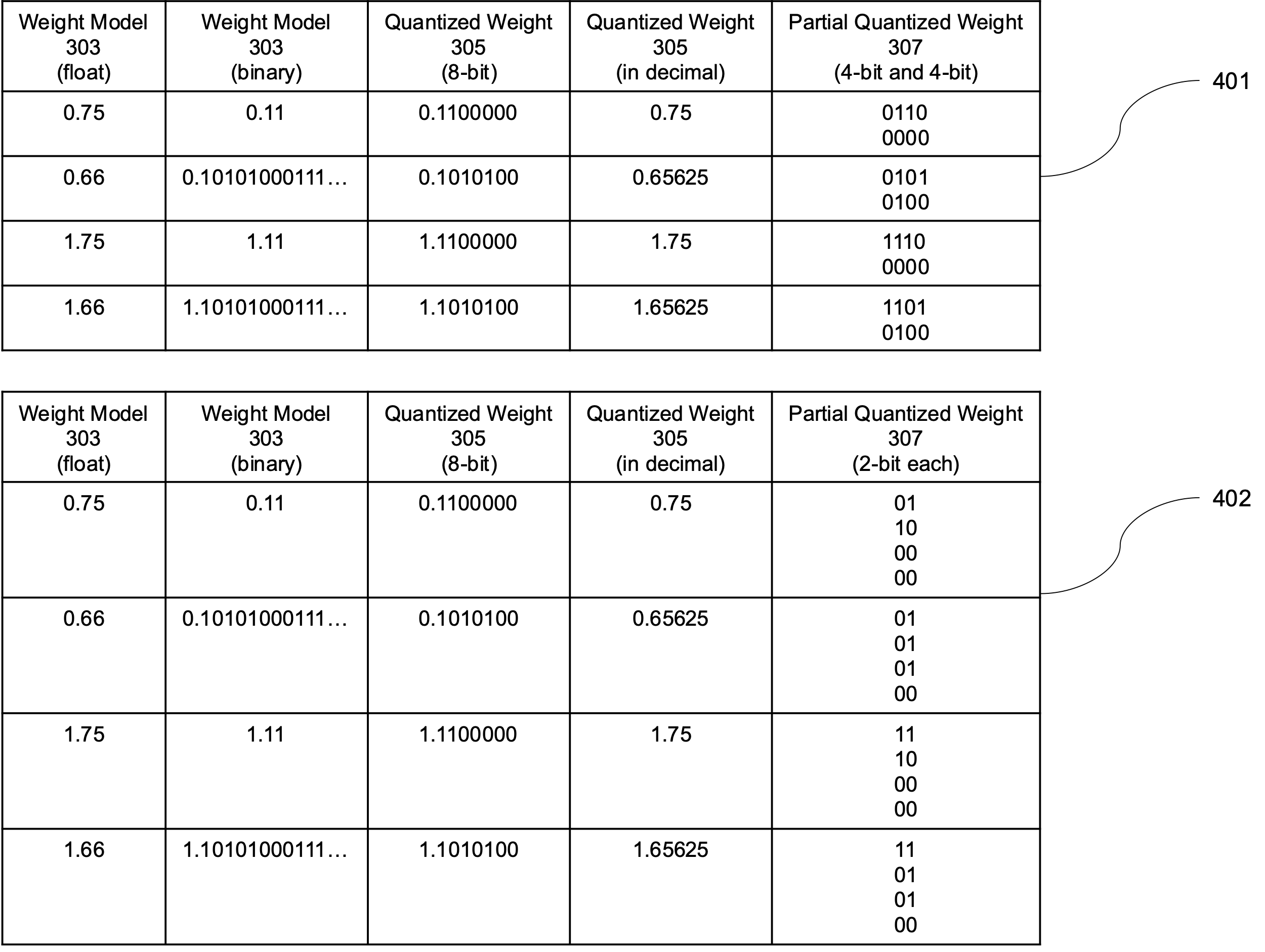
**Fig. 1:** **An exemplary of conventional neuromorphic inference chip**



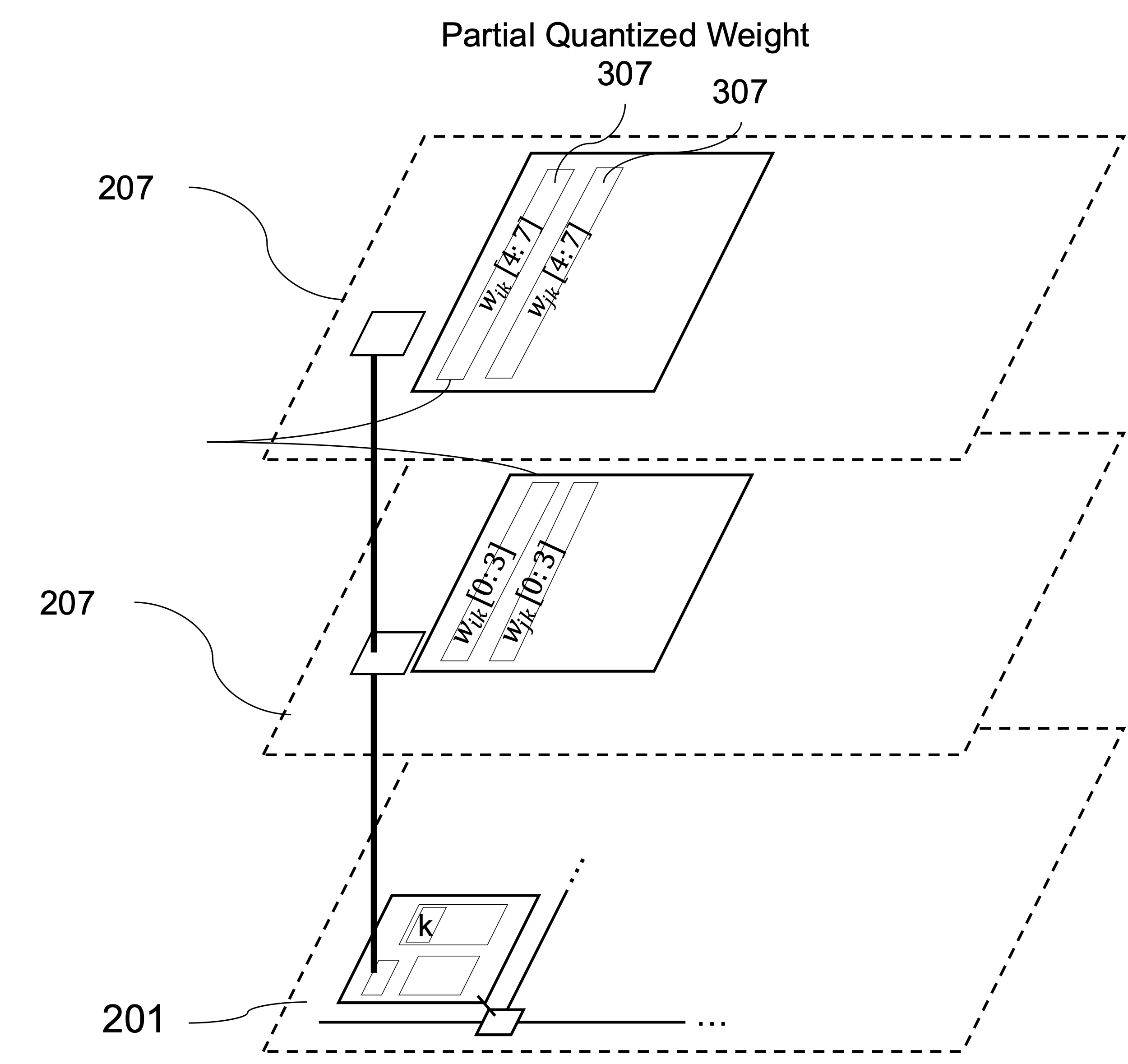
**Fig. 2: An exemplary of the proposed neuromorphic inference chip.**



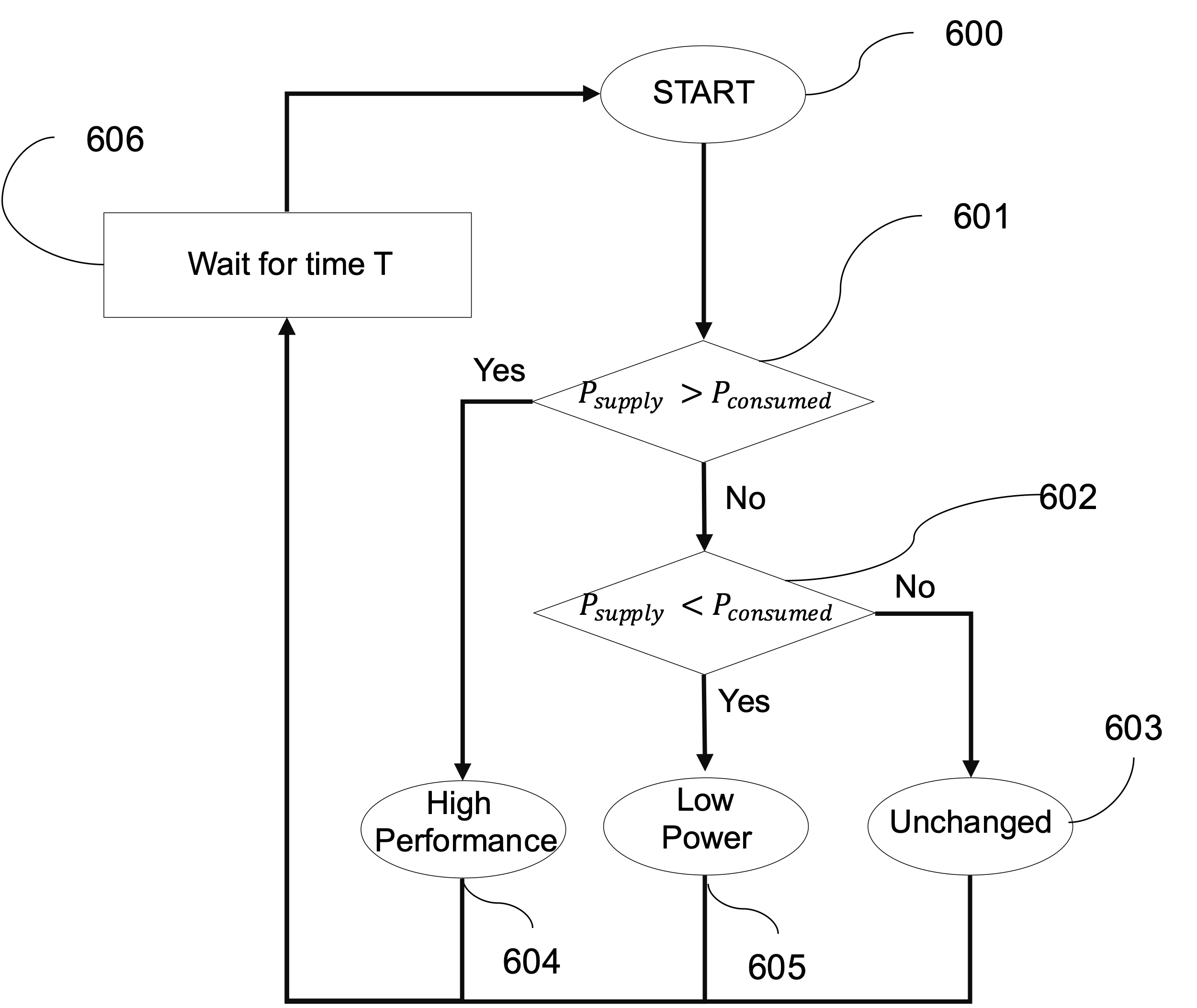
**Fig. 3: The dynamic quantization process.**



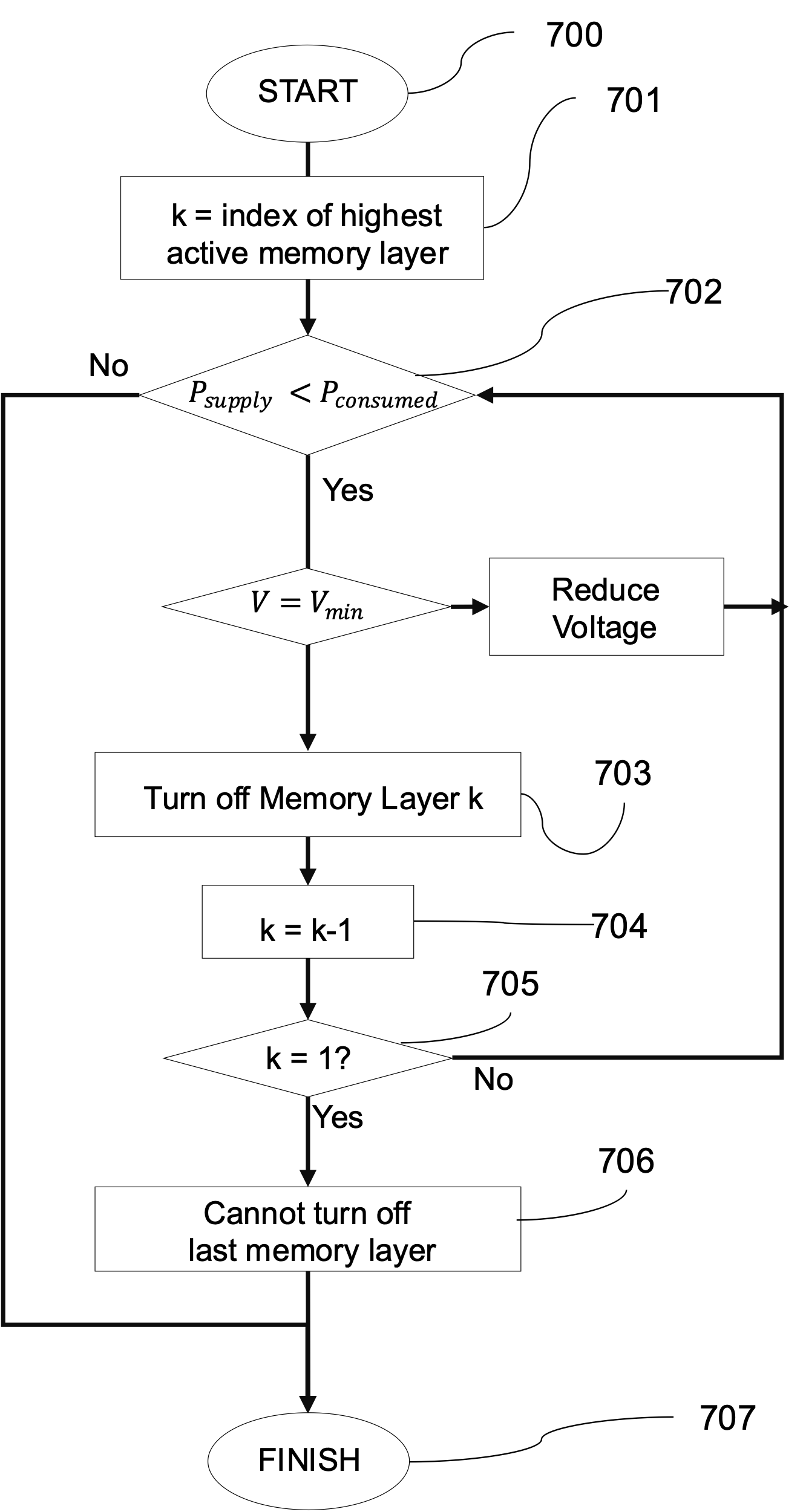
**Fig. 4: Examples of dynamic quantization.**



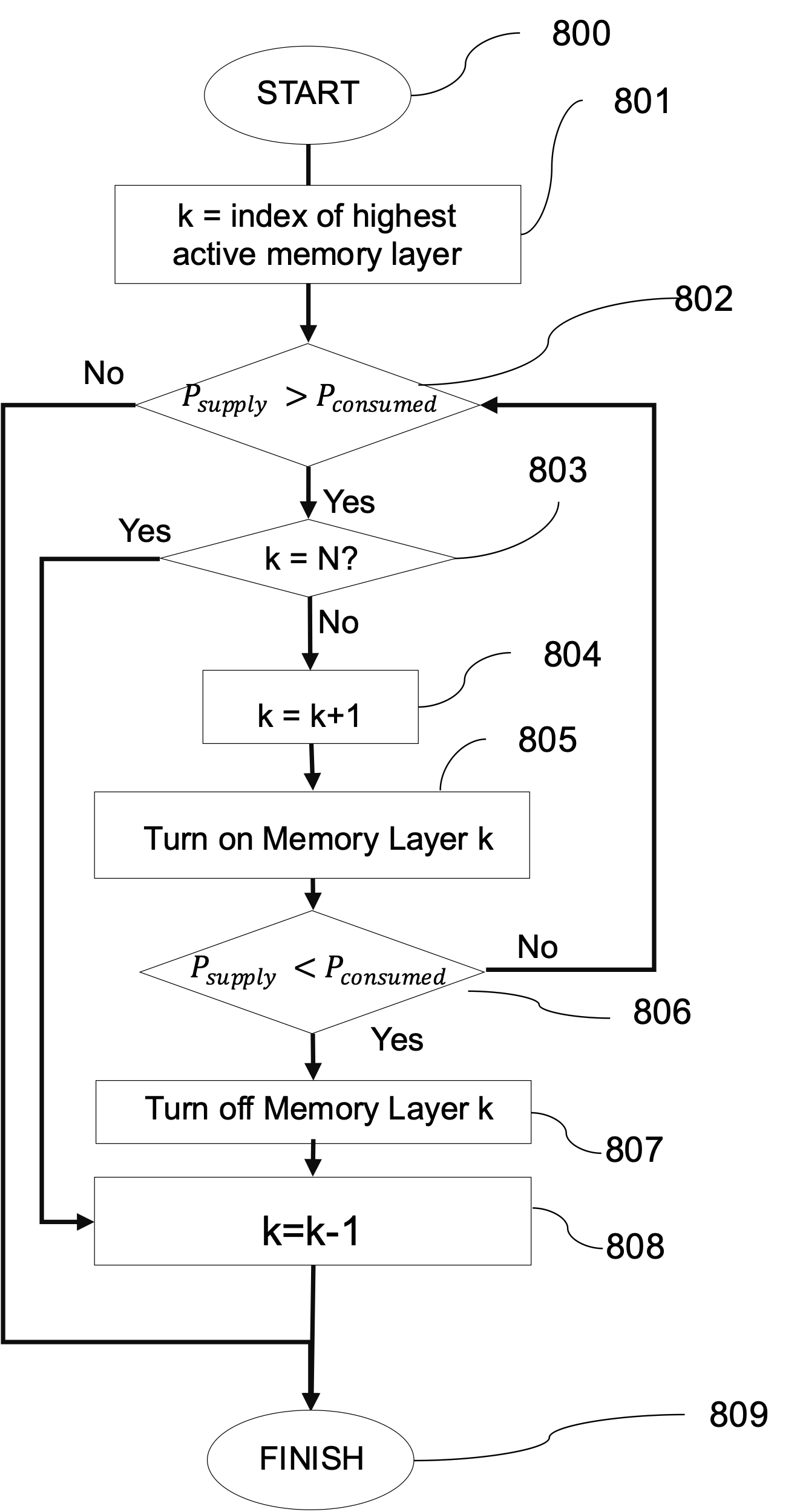
**Fig. 5: Mapping location of partial quantized weight.**

****

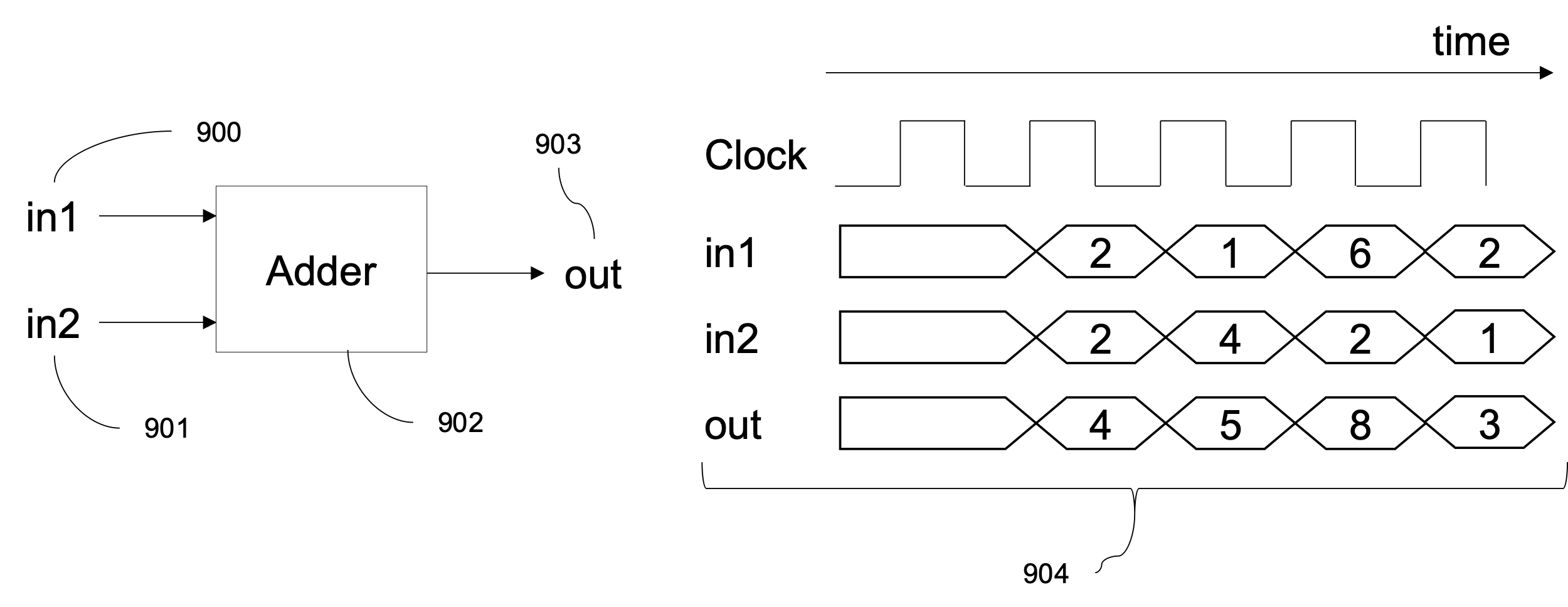
**Fig. 6: Mode decision flow-chart.**

****

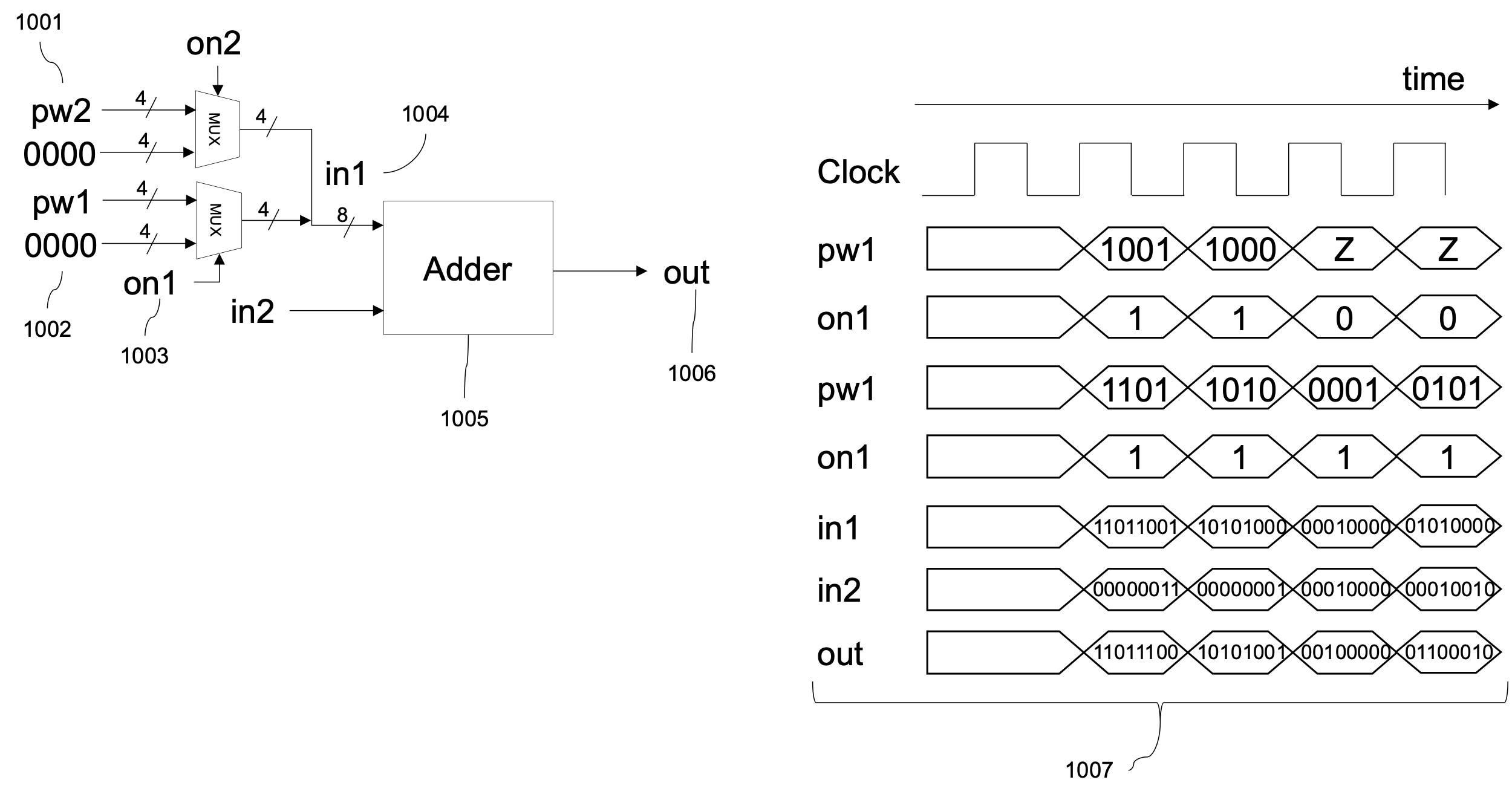
**Fig. 7: Flow low-power mode (605).**

****

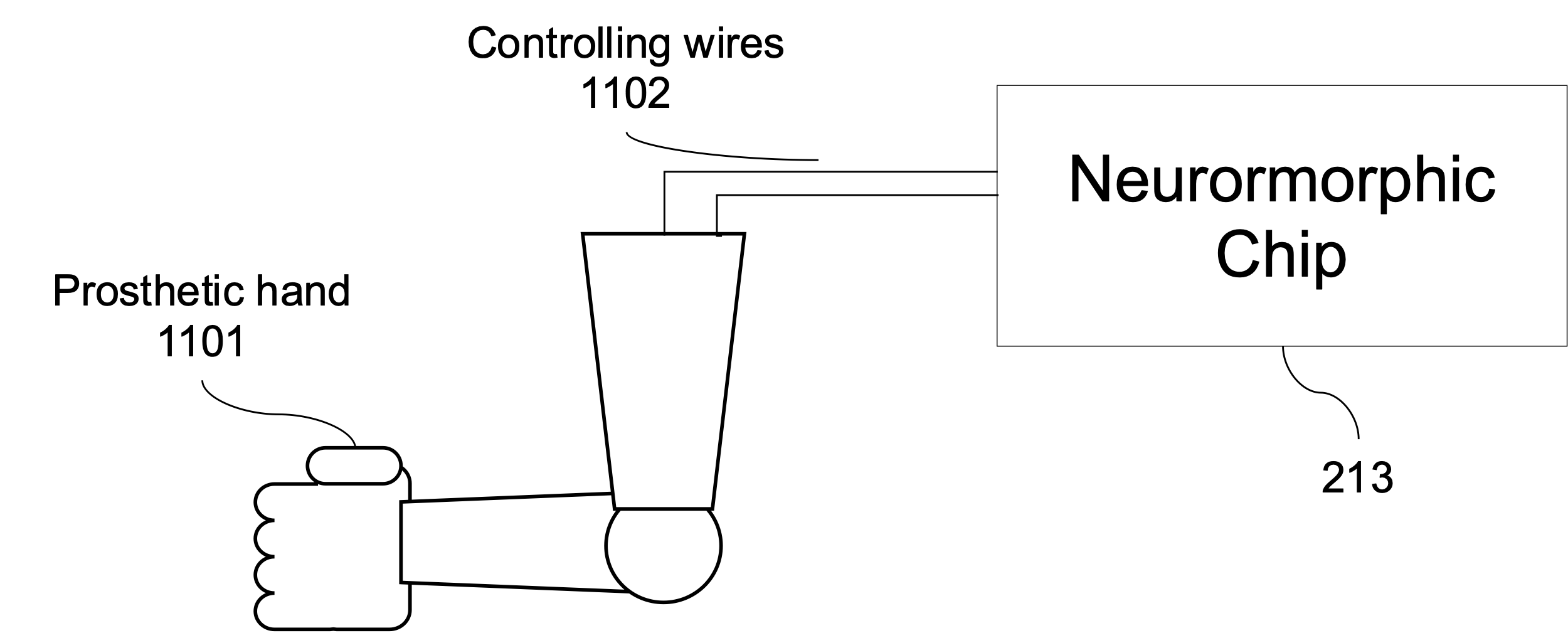
**Fig. 8: Flow of high-performance mode (604).**

****

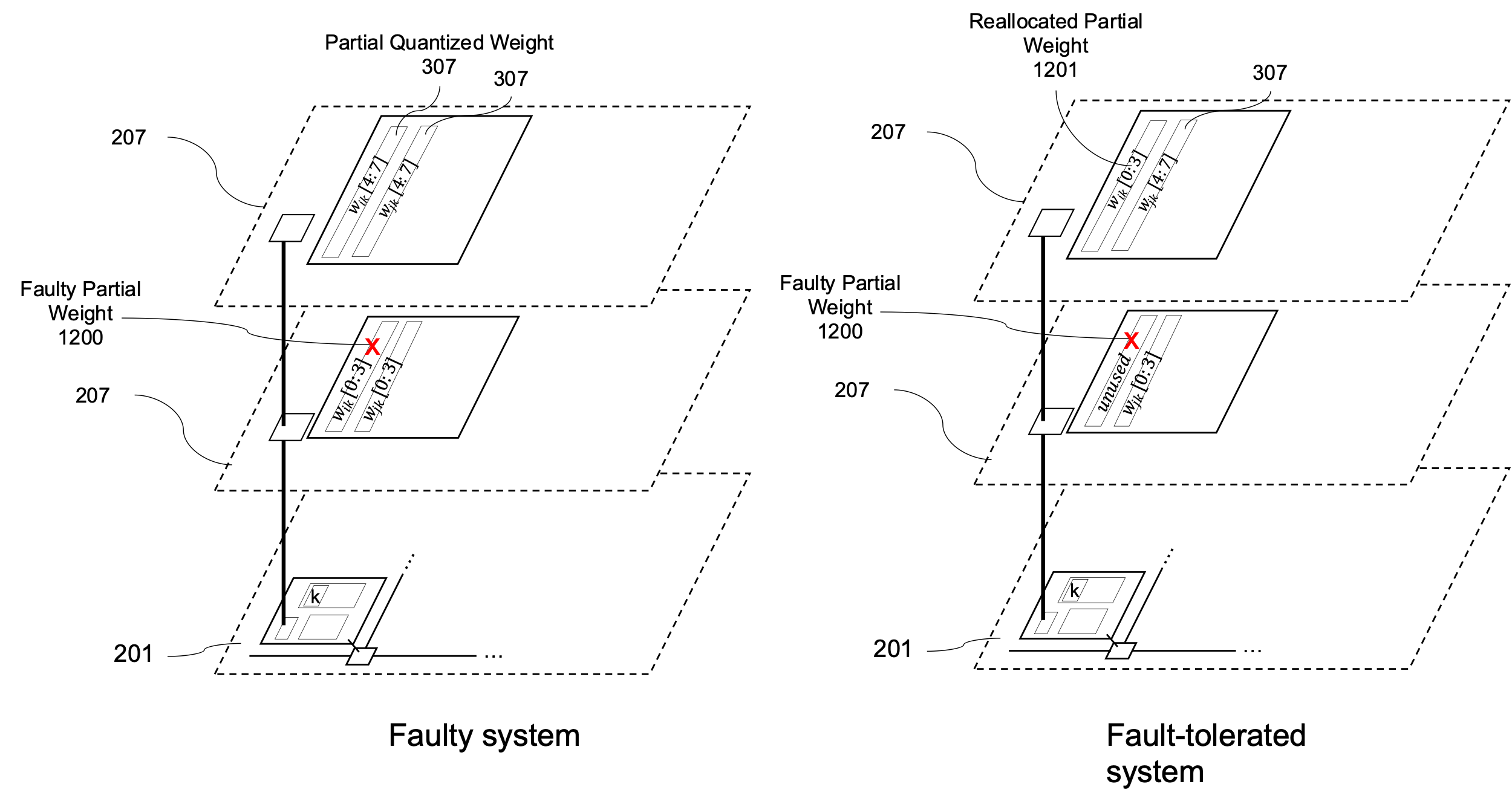
**Fig. 9: Conventional Adder.**

****

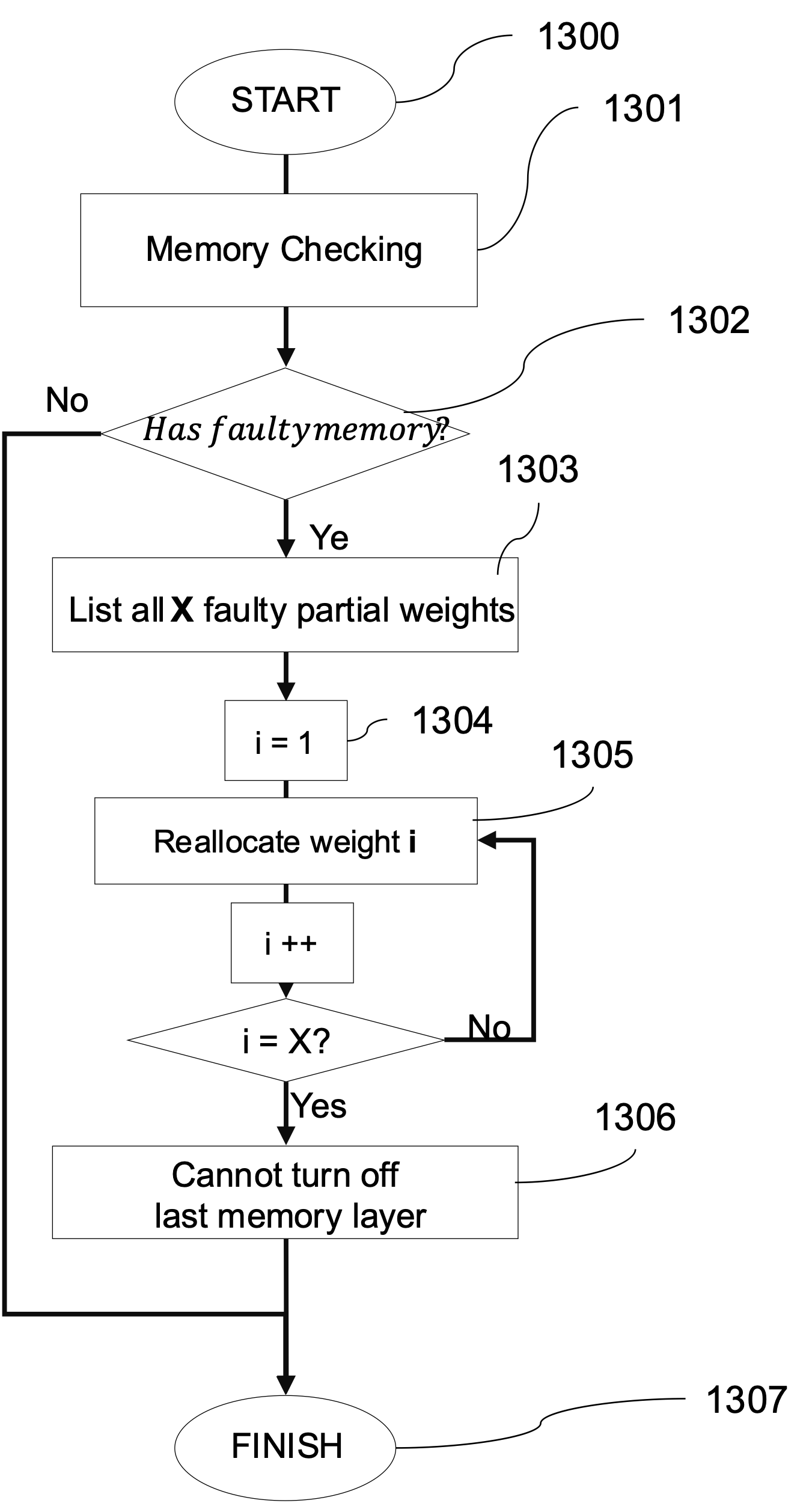
**Fig. 10: Modified Adder to support turning off layers.**

****

**Fig. 11: Prosthetic hand controlling system.**

****

**Fig. 12: Reallocating partial weights for fault-tolerant Prosthetic hand controlling system.**

****

**Fig. 13: Flow-chart for Reallocating partial weights for fault-tolerant Prosthetic hand controlling system.**

REFERENCES

[1] A. Balaji et al., “Mapping spiking neural networks to neuromorphic hardware,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 1, pp. 76–86, 2019.

[2] Y. Ji et al., "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints", The 49th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 1-13, 2016.

[3] Adarsha Balaji, Thibaut Marty, Anup Das, and Francky Catthoor. 2020. Run-time Mapping of Spiking Neural Networks to Neuromorphic Hardware. J. Signal Process. Syst. 92, 11 (Nov 2020), 1293–1302.

[4] Jin, X. (2010). Parallel simulation of neural networks on spinnaker universal neuromorphic hardware, PhD Thesis, (The University of Manchester (United Kingdom))

[5] Spyrou, T., El-Sayed, S. A., Afacan, E., Camuñas-Mesa, L. A., Linares-Barranco, B., & Stratigopoulos, H. G. (2021, February). Neuron fault tolerance in spiking neural networks. In 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 743-748). IEEE.

[6] Putra, R. V. W., Hanif, M. A., & Shafique, M. (2021, November). ReSpawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) (pp. 1-9). IEEE.

[7] A. Namazi et al., “A majority-based reliability-aware task mapping in high-performance homogenous NoC architectures,” ACM Transactions on Embedded Computing Systems (TECS), vol. 17, no. 1, pp. 1–31, 2017.

[8] P. K. Sahu et al., “Application mapping onto mesh-based networkon-chip using discrete particle swarm optimization,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 2, pp. 300–312, 2013

[9] Edmonds, Jack; Karp, Richard M. (1972). "Theoretical improvements in algorithmic efficiency for network flow problems" (PDF). Journal of the ACM. 19 (2): 248–264. doi:10.1145/321694.321699.

[10] Sawada, J., Modha, D.S., Arthur, J.V., Cassidy, A.S., Datta, P., Appuswamy, R., Nayak, T.K., Taba, B.K., Otero, C.O., Akopyan, F. and Amir, A., 2022. Modular neural network computing apparatus with distributed neural network storage. U.S. Patent Application 17/077,720.

[11] Appuswamy, Rathinakumar, John V. Arthur, Andrew S. Cassidy, Pallab Datta, Steven K. Esser, Myron D. Flickner, Jennifer Klamo et al. "Massively parallel neural inference computing elements." U.S. Patent 10,621,489, issued April 14, 2020.

[12] Rivera, Rodrigo Alvarez Icaza, John V. Arthur, Andrew S. Cassidy, Bryan L. Jackson, Paul A. Merolla, Dharmendra S. Modha, and Jun Sawada. "Scaling multi-core neurosynaptic networks across chip boundaries." U.S. Patent 9,924,480, issued March 20, 2018.

[13] Davies, Michael I. "Scalable neuromorphic core with shared synaptic memory and variable precision synaptic memory." U.S. Patent 10,824,937, issued November 3, 2020.