2024 Spring Machine Learning Intelligent Chip Design

Final Project Report

313510235 陳孟頡

Part I. Simulation Result

Simulation result of dog.txt

Top 100 classes:

idx	val	possibility	class name
207	16.59	38.63	golden retriever
175	15.57	13.86	otterhound
220	15.36	11.26	Sussex spaniel
163	15.00	7.86	bloodhound
219	14.59	5.22	cocker spaniel
168	14.39	4.28	redbone
160	14.35	4.07	Afghan hound
213	14.18	3.46	Irish setter
291	14.10	3.19	lion
211	13.01	1.07	vizsla
244	12.81	0.88	Tibetan mastiff
216	12.69	0.78	clumber
200	12.46	0.62	Tibetan terrier
159	12.42	0.59	Rhodesian ridgeback
152	12.38	0.57	Japanese spaniel
167	12.01	0.39	English foxhound
208	11.65	0.28	Labrador retriever
294	11.63	0.27	brown bear
165	11.51	0.24	black-and-tan coonhound
262	11.46	0.23	Brabancon griffon
156	11.40	0.21	Blenheim spaniel
185	11.29	0.19	Norfolk terrier
260	11.11	0.16	chow
267	11.08	0.16	standard poodle
365	10.87	0.13	orangutan
154	10.74	0.11	Pekinese
368	10.70	0.11	gibbon
215	10.66	0.10	Brittany spaniel
161	10.59	0.10	basset
214	10.47	0.08	Gordon setter
176	10.39	0.08	Saluki
212	10.31	0.07	English setter
852	9.88	0.05	tennis ball
255	9.87	0.05	Leonberg
194	9.77	0.04	Dandie Dinmont
672	9.59	0.04	mountain tent
274	9.44	0.03	dhole
335	9.44	0.03	fox squirrel
371	9.40	0.03	patas
190	9.31	0.03	Sealyham terrier
384	9.29	0.03	indri
193	9.15	0.02	Australian terrier
166	9.00	0.02	Walker hound
206	8.92	0.02	curly-coated retriever
380	8.90	0.02	titi
170	8.86	0.02	Irish wolfhound
191	8.85	0.02	Airedale
379	8.84	0.02	howler monkey
243	8.77	0.02	bull mastiff

Simulation result of cat.txt

Top 100 classes:

idx	val	possibility	class name
285	20.21	96.38	Egyptian cat
281	16.14	1.65	tabby
282	15.73	1.10	tiger cat
287	14.79	0.43	lynx
728	14.41	0.29	plastic bag
330	12.73	0.05	wood rabbit
331	12.19	0.03	hare
457	10.94	0.01	bow tie
335	10.67	0.01	fox squirrel
463	10.57	0.01	bucket
478	10.32	0.00	carton
876	10.29	0.00	tub
622	10.18	0.00	lens cap
904	10.01	0.00	window screen
700	9.56	0.00	paper towel
278	9.39	0.00	kit fox
8	9.29	0.00	hen
284	9.21	0.00	Siamese cat
722	8.80	0.00	ping-pong ball
434	8.77	0.00	bath towel
452	8.71	0.00	bonnet
289	8.31	0.00	snow leopard
753	8.30	0.00	radiator
681	8.11	0.00	notebook
673	8.04	0.00	mouse
840	8.03	0.00	swab
773	7.94	0.00	saltshaker
782	7.90	0.00	screen
896	7.71	0.00	washbasin
44	7.70	0.00	alligator lizard
719	7.69	0.00	piggy bank
138	7.68	0.00	bustard
998	7.57	0.00	ear
543	7.52	0.00	dumbbell
451	7.49	0.00	bolo tie
283	7.40	0.00	Persian cat
272	7.32	0.00	coyote
999	7.27	0.00	toilet tissue
769	7.23	0.00	rule
713	7.14	0.00	photocopier
760	7.12	0.00	refrigerator
474	7.08	0.00	cardigan
534	6.99	0.00	dishwasher
797	6.94	0.00	sleeping bag
552	6.93	0.00	feather boa
793	6.93	0.00	shower cap
322	6.92	0.00	ringlet
288	6.91	0.00	leopard
750	6.91	0.00	quilt

189	8.59	0.01	Lakeland terrier
218	8.57	0.01	Welsh springer spaniel
182	8.53	0.01	Border terrier
187	8.53	0.01	Yorkshire terrier
209	8.51	0.01	Chesapeake Bay retriever
273	8.37	0.01	dingo
222	8.32	0.01	kuvasz
226	8.29	0.01	briard
382	8.23	0.01	squirrel monkey
169	8.18	0.01	borzoi
373	8.17	0.01	macaque
256	8.12	0.01	Newfoundland
155	8.09	0.01	Shih-Tzu
186	8.04	0.01	Norwich terrier
299	7.83	0.01	meerkat
184	7.82	0.01	Irish terrier
164	7.74	0.01	bluetick
162	7.70	0.01	beagle
292	7.63	0.00	tiger
266	7.62	0.00	miniature poodle
298	7.44	0.00	mongoose
231	7.40	0.00	collie
257	7.33	0.00	Great Pyrenees
158	7.33	0.00	toy terrier
181	7.31	0.00	Bedlington terrier
188	7.28	0.00	wire-haired fox terrier
265	7.28	0.00	toy poodle
247	7.25	0.00	Saint Bernard
264	7.23	0.00	Cardigan
221	7.23	0.00	Irish water spaniel
463	7.17	0.00	bucket
101	7.14	0.00	tusker
180	7.08	0.00	American Staffordshire terrier
238	7.07	0.00	Greater Swiss Mountain dog
201	7.03	0.00	silky terrier
354	7.03	0.00	Arabian camel
339	6.99	0.00	sorrel
204	6.89	0.00	Lhasa
235	6.88	0.00	German shepherd
183	6.79	0.00	Kerry blue terrier
271	6.77	0.00	red wolf
286	6.66	0.00	cougar
227	6.66	0.00	kelpie
370	6.62	0.00	guenon
252	6.60	0.00	affenpinscher
372	6.52	0.00	baboon
377	6.51	0.00	marmoset
240	6.49	0.00	Appenzeller
202	6.48	0.00	soft-coated wheaten terrier
151	6.48	0.00	Chihuahua
369	6.43	0.00	siamang

897	6.84	0.00	washer
36	6.83	0.00	terrapin
326	6.76	0.00	lycaenid
723	6.75	0.00	pinwheel
7	6.72	0.00	cock
333	6.66	0.00	hamster
519	6.65	0.00	crate
588	6.65	0.00	hamper
883	6.54	0.00	vase
725	6.50	0.00	pitcher
438	6.46	0.00	beaker
469	6.37	0.00	caldron
350	6.36	0.00	ibex
968	6.28	0.00	cup
280	6.25	0.00	grey fox
290	6.22	0.00	jaguar
83	6.18	0.00	prairie chicken
680	6.15	0.00	nipple
41	6.13	0.00	whiptail
572	6.13	0.00	goblet
742	6.07	0.00	printer
359	6.03	0.00	black-footed ferret
353	6.01	0.00	gazelle
937	5.96	0.00	broccoli
862	5.94	0.00	torch
286	5.91	0.00	cougar
504	5.86	0.00	coffee mug
855	5.81	0.00	thimble
332	5.74	0.00	Angora
400	5.73	0.00	academic gown
845	5.69	0.00	syringe
38	5.68	0.00	banded gecko
271	5.63	0.00	red wolf
357	5.61	0.00	mink
47	5.60	0.00	African chameleon
869	5.58	0.00	trench coat
584	5.57	0.00	hair slide
298	5.45	0.00	mongoose
761	5.39	0.00	remote control
470	5.38	0.00	candle
636	5.36	0.00	mailbag
620	5.35	0.00	laptop
268	5.31	0.00	Mexican hairless
273	5.31	0.00	dingo
514	5.28	0.00	cowboy boot
596	5.25	0.00	hatchet
885	5.24	0.00	velvet
929	5.23	0.00	ice lolly
899	5.22	0.00	water jug
473	5.20	0.00	can opener
377	5.15	0.00	marmoset

Part II. Implementation

以下概略說明 AXI4 implementation 以及整體架構 optimization，由於 final project 中使用的是唯獨記憶體(ROM)，因此作業針對 read channel 進行實作：

		#	address
conv1	weight	23232	0
	bias	64	23232
conv2	weight	307200	23296
	bias	192	330496
conv3	weight	663552	330688
	bias	384	994240
conv4	weight	884736	994624
	bias	256	1879360
conv5	weight	589824	1879616
	bias	256	2469440
input		150528	2469696
fc6	weight	37748736	2620224
	bias	4096	40368960
fc7	weight	16777216	40373056
	bias	4096	57150272
fc8	weight	4096000	57154368
	bias	1000	61250368

1. AXI4 Architecture

(1) Signals

***Address Channel**

ARID[3:0]:

用來識別不同的 master 資料請求，本架構由於採用 controller 與 ROM 一一對應，因此沒有用到 AWID signal。

ARADDR[31:0]:

為了模擬真實 memory，將不同的資料放在 ROM 當中，並以 address 進行讀取。ROM 資料擺放位址如右圖:

ARLEN[7:0]:

每一次 access ROM 的 beat 數，由於時間因素，此次作業並無針對 ARLEN 進行實作。

ARSIZE[2:0]:

每一個 beat 所傳輸的 bit 數，為了方便控制，同時達到提高傳輸速度效果，此程式支援到 128 bits(一次 4 個 float)。

```
int data_per_read;
switch (BURST_SIZE) {
    case 2:  { data_per_read = 1; break; }      // 4  bytes
    case 3:  { data_per_read = 2; break; }      // 8  bytes
    case 4:  { data_per_read = 4; break; }      // 16 bytes
    default: { data_per_read = 1; break; }
}
```

ARBURST[1:0]:

在不同的應用中，可使用不同的 burst mode 方便資料存取，此次 ROM 為 address 以 ARSIZE 對應的 byte 遞增，因此實作以 INCR 模式為主。

ARVALID:

表示 ARID、ARADDR 等 address channel 資料有效，主要用在 handshake。

```
if (ARVALID.read()) {
    ARREADY.write(true);    // tell master get the request

    read_addr = ARADDR.read().to_uint();
    master_id = ARID.read().to_uint();

    read_leng = ARLEN.read().to_uint();     // read 8 bits burst length
    read_size = ARSIZE.read().to_uint();    // read 3 bits burst size
    read_type = ARBURST.read().to_uint();   // read 2 bits burst type

    rom_cs = ROM_PROCESS;
}
```

ARREADY:
ROM 回覆 controller 收到請求，準備開始傳送讀取資料。

***Data Channel**
RID[3:0]:
回傳資料的識別碼，對應原先讀取 controller 請求中的 ARID。同上述，此次實作的 controller 與 ROM 為一一對應，因此 ARID 固定為 0。

RDATA:
實際的讀取資料。為了模擬 AXI4 中一次傳輸多 bit 資料，此次使用 RDATA[127:0]進行 ROM 資料傳輸，最多一次可以傳送 128/32 = 4 個 float 數值，增加 ROM throughput。

```
while (infile >> temp_read[temp_index]) {
    temp_index++;

    // When we've read enough data for one 128-bit word
    if (temp_index == data_per_read) {
        sc_lv<128> temp;

        // Initialize unused elements to 0
        for (int i = data_per_read; i < 4; i++) {
            temp_read[i] = 0.0f;
        }

        // Convert all 4 floats to sc_lv<128>
        for (int i = 0; i < 4; i++) {
            uint32_t float_bits = *reinterpret_cast<uint32_t*>(&temp_read[i]);
            temp.range(31 + i*32, i*32) = float_bits;
        }

        data_vec.push_back(temp);
        temp_index = 0;  // Reset for next read cycle
    }
}
```

RLAST:
一次 burst 的結束，實作中用來控制 controller FSM，從讀取回到等待下一筆資料模式。

```
if (RLAST.read()) {
    cout << "Time = " << setw(14) << sc_time_stamp()

    if(!get_weight[current_layer]) {
        get_weight[current_layer] = true;
    }
    else if (!get_bias[current_layer]) {
        get_bias[current_layer] = true;
        current_layer++;
    }

    c0_rx_cs = CON0_RX_WAIT;
}
```

RVALID:

代表傳輸資料有效，為了方便實作，同時加快讀取資料速度，本次作業在傳送時會將 RVALID 持續拉高，以方便連續讀取資料。

```
if (RVALID.read()) {
    sc_lv<128> temp_read = RDATA.read();
    extract_rom_data(temp_read, current_layer, get_weight[current_layer], get_bias[current_layer]);
```

RREADY:

Controller 會持續將 RREADY 拉高，告訴 ROM 處於可以讀取資料狀態。

(2) **Function**

以下為統整各個 ROM 所實作出的功能：

```
enum ROM_State {
    ROM_IDLE,
    ROM_PROCESS,
    ROM_READ,
    ROM_WAIT,
    ROM_RESPONSE,
    ROM_DONE
};
```

a. FSM Behavior:

為了讓 ROM 更接近實際硬體運作，使用 FSM 進行實作，流程為：IDLE -> 收到讀取請求 -> 處理 address 資料 -> 輸出讀取資料 -> 完成讀取回到 IDLE。

b. Read Data with Address:

如在 (1) signals 中所述，使用 address 讀取對應 weight, bias, input 資料。為了減少程式執行所需的記憶容量，並不會將所有資料存於 ROM 中，而是在讀到對應 address 時開啟檔案，依序讀入所需資料。

```
string ROM::open_file(uint32_t addr) {
    if (addr < CONV1_B_ADDR)    return "./data/conv1_weight.txt";
    if (addr < CONV2_W_ADDR)    return "./data/conv1_bias.txt";
    if (addr < CONV2_B_ADDR)    return "./data/conv2_weight.txt";
    if (addr < CONV3_W_ADDR)    return "./data/conv2_bias.txt";
    if (addr < CONV3_B_ADDR)    return "./data/conv3_weight.txt";
    if (addr < CONV4_W_ADDR)    return "./data/conv3_bias.txt";
    if (addr < CONV4_B_ADDR)    return "./data/conv4_weight.txt";
    if (addr < CONV5_W_ADDR)    return "./data/conv4_bias.txt";
    if (addr < CONV5_B_ADDR)    return "./data/conv5_weight.txt";
    if (addr < INPUT_ADDR)      return "./data/conv5_bias.txt";
    if (addr < FC6_W_ADDR)      return "./data/" + IMAGE_FILE_NAME;
    if (addr < FC6_B_ADDR)      return "./data/fc6_weight.txt";
    if (addr < FC7_W_ADDR)      return "./data/fc6_bias.txt";
    if (addr < FC7_B_ADDR)      return "./data/fc7_weight.txt";
    if (addr < FC8_W_ADDR)      return "./data/fc7_bias.txt";
    if (addr < FC8_B_ADDR)      return "./data/fc8_weight.txt";
    if (addr < MAX_ADDR)        return "./data/fc8_bias.txt";

    return "";
}
```

c. Handshake:

ARVALID vs. ARREADY

```
if (ARVALID.read()) {
    ARREADY.write(true);    // tell master get the request

    read_addr = ARADDR.read().to_uint();
    master_id = ARID.read().to_uint();

    read_leng = ARLEN.read().to_uint();     // read 8 bits burst length
    read_size = ARSIZE.read().to_uint();    // read 3 bits burst size
    read_type = ARBURST.read().to_uint();   // read 2 bits burst type

    rom_cs = ROM_PROCESS;
}
```

RVALID vs. RREADY

```
if(RREADY.read()) {
    // cout << "Time = " << setw(14) <<

    RID.write(master_id);
    RDATA.write(data_vec[read_index]);
    RRESP.write(0);
    RLAST.write(false);
    RVALID.write(true);

    rom_cs = ROM_READ;
}
```

d. Burst Size:

支援不同的傳輸 burst size。

```
int data_per_read;
switch (r_size) {
    case 0:  { data_per_read = 0; break; }      // 1  byte
    case 1:  { data_per_read = 0; break; }      // 2  bytes
    case 2:  { data_per_read = 1; break; }      // 4  bytes
    case 3:  { data_per_read = 2; break; }      // 8  bytes
    case 4:  { data_per_read = 4; break; }      // 16 bytes
    default: { data_per_read = 1; break; }
}
```

e. Read Delay

使用一般 memory 時，會有 read delay，即在 ROM 收到請求後所需等待的 cycle，此處已倒數形式實作。

```
if (read_delay <= 0) {

    rom_cs = ROM_WAIT;
}
else {
    read_delay--;
}
```

f. ROM Response:

ROM 會根據 address 是否合理、是否正常開啟 weight, bias 檔案等，將讀取狀態透過 RRESP 讓 controller 知道，以方便後續處理。

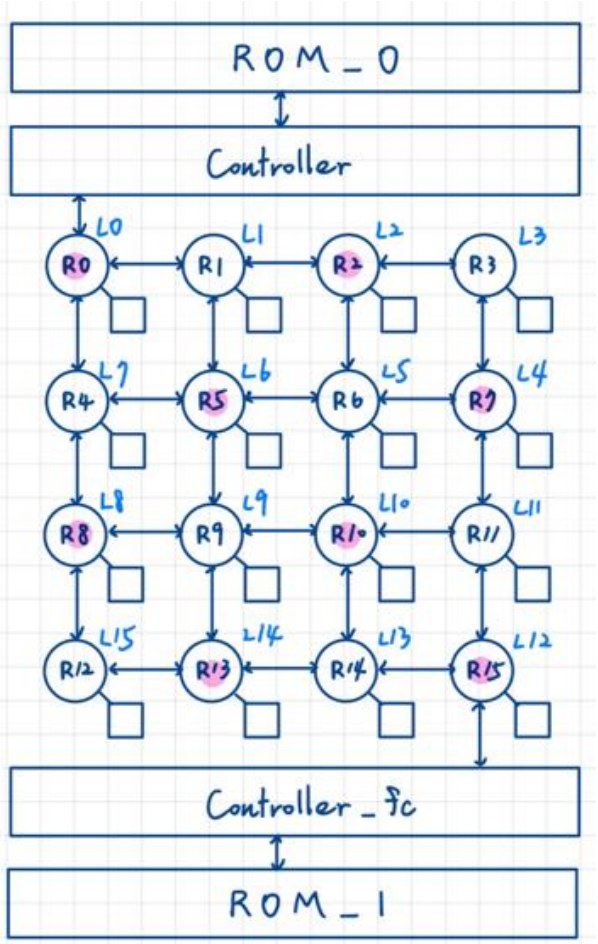
```
string input_file = open_file(read_addr);
if (input_file == "") {
    cout << "Time = " << setw(14) << sc_time_stamp() << "  [ERROR]  ROM-" << rom_id << "
    read_resp = 3;
}

ifstream fin;
fin.open(input_file.c_str());
if (!fin.is_open()) {
    cout << "Time = " << setw(14) << sc_time_stamp() << "  [ERROR]  ROM-" << rom_id << "
    read_resp = 2;
}
else {
    cout << "Time = " << setw(14) << sc_time_stamp() << "  ROM-" << rom_id << "        (
}

process_read_data(read_size, fin);
fin.close();

if(data_vec.empty()) {
    cout << "Time = " << setw(14) << sc_time_stamp() << "  [ERROR]  ROM-" << rom_id << "
    read_resp = 2;
}

rom_cs = ROM_WAIT;
```

2. Optimization

以下針對此次設計優化部分進行說明：

(1) Architecture

在 HW4 中，觀察到 fully connect 層的讀取及傳送佔了大部分的執行時間，因此將其獨立為另外一個 ROM，並以額外的 controller 進行控制如此便能平行處理兩邊資料。而為了符合實際設計時的配置，此處將 convolution 以及 input 資料放在 ROM 0，fully connect 層資料放在 ROM 1 中，並封別接上 router 0, router 12，架構如下圖：



各個 core 與其對應 AlexNet 層數與 Hw4 相同：

Computation Order	Core ID	Calculation
1	0	Convolution 1
2	1	ReLU 1 + Max-Pooling 1
3	2	Convolution 2
4	3	ReLU 2 + Max-Pooling 2
5	7	Convolution 3
6	6	ReLU 3
7	5	Convolution 4
8	4	ReLU 4
9	8	Convolution 5
10	9	ReLU 5 + Max-Pooling 5
11	10	Fully Connected 6
12	11	ReLU 6
13	15	Fully Connected 7
14	14	ReLU 7
15	13	Fully Connected 8
16	12	Softmax

就執行結果而言，採用此架構確實加快總執行速度，但由於 bottleneck 的 FC6 所需 cycle 數仍非常多，因此加速比例並不明顯。若要近一步加速，應嘗試將 FC6 獨立於一個 ROM。

(2) Data Reading and Sending Scheduling

在原先 HW4 中，為確保正確接收 weight, bias，controller 在接收完所有資料才會開始發送至各個 core。在此 optimized 版本中，將其改為收完即送，透過 get_weight 以及 get_bias 兩個 flag，當一層的 weight 或 bias 完成接收後，即可發送給 core，以減少中間等待時間。

具體做法為將原先一個 controller FSM 改為 transmit FSM 與 receive FSM 同時進行控制，receive FSM 接收完會將 flag 拉起，transmit FSM 看到後即可開始傳送。

Rx FSM

```
if (RLAST.read()) {
    cout << "Time = " << setw(14) << sc_ti

    if(!get_weight[current_layer]) {
        get_weight[current_layer] = true;
    }
    else if (!get_bias[current_layer]) {
        get_bias[current_layer] = true;
        current_layer++;
    }

    c0_rx_cs = CON0_RX_WAIT;
}
```

Tx FSM

```
if (get_weight[current_send] && !send_weight) {
    cout << "Time = " << setw(14) << sc_time_stamp()

    make_controller_flits(0, current_send);

    cout << temp_flits.size() << " flits" << endl;

    req_tx.write(true);
    flit_tx.write(temp_flits[flit_index]);

    c0_tx_cs = CON0_TX_WAIT;
}
else if (get_bias[current_send]) {
    cout << "Time = " << setw(14) << sc_time_stamp()

    make_controller_flits(1, current_send);

    cout << temp_flits.size() << " flits" << endl;

    req_tx.write(true);
    flit_tx.write(temp_flits[flit_index]);

    c0_tx_cs = CON0_TX_WAIT;
}
```