KAN Kolmogorov Arnold Network Note

研究任務

1. 閱讀 KAN Kolmogorov-Arnold Network 論文
2. 設計 KAN 網路系統階層式架構 IDEF0
3. 設計 KAN 網路系統每個功能模組離散事件建模 Grafcet
4. 以 MIAT 方法論合成每個 Grafcet 控制器電路
5. 以 ChatGPT 合成每個 Grafcet Datapath 電路
6. FPGA 整合驗證

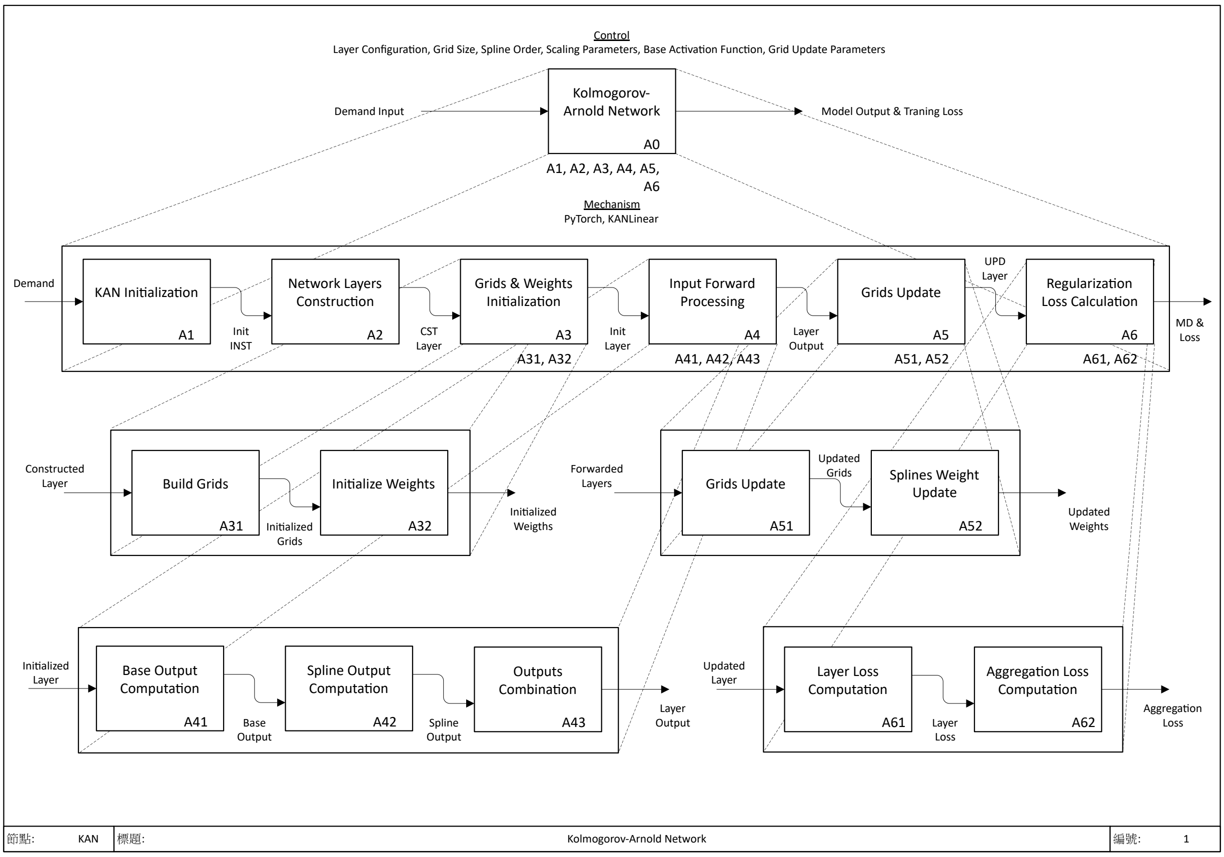
> ⚠ **Attention**
>
> Kolmogorov-Arnold Network 之研究任務須於六月底完成，作為暑期實習前置條件。

> ✎ **Seealso**
>
> 目標預定之所有 (第一到第六點) 研究任務，當前初版皆已完成，接下來主要目標為模型量化和優化相關電路設計及輸出結果。
>
> - Integer Quantization 部分已經完成

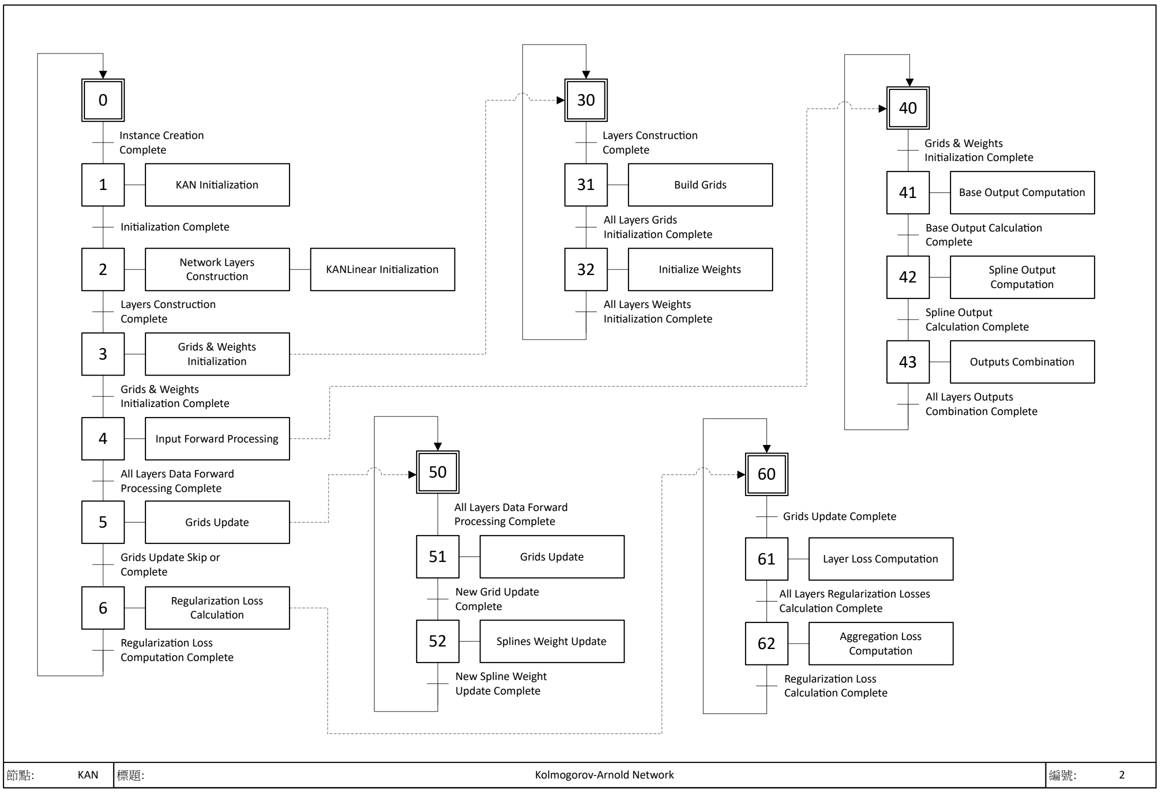
開放原始碼參考

- Code: https://github.com/KindXiaoming/pykan
- Reference: https://arxiv.org/abs/2404.19756

設計階層式架構 IDEF0



設計功能模組離散事件建模 Grafcet



Python 模擬驗證

- 基於 IDEF0 和 Grafcet 重構後之 Kolmogorov-Arnold Network (PyTorch)

```python
import math

import torch
import torch.nn.functional as F


class KANLinear(torch.nn.Module):
    def __init__(
            self,
            in_features,
            out_features,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            enable_standalone_scale_spline=True,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KANLinear, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.grid_size = grid_size
        self.spline_order = spline_order

        # 構建網格點
        self.grid = self.build_grid(grid_range, grid_size, spline_order)

        # 初始化基礎權重和樣條權重
        self.base_weight, self.spline_weight, self.spline_scaler = self.initialize_weights(
```

```
            out_features, in_features, grid_size, spline_order, scale_base, scale_spline,
enable_standalone_scale_spline
        )

        self.scale_base = scale_base
        self.scale_spline = scale_spline
        self.enable_standalone_scale_spline = enable_standalone_scale_spline
        self.base_activation = base_activation()
        self.grid_eps = grid_eps

    def build_grid(self, grid_range, grid_size, spline_order):
        h = (grid_range[1] - grid_range[0]) / grid_size
        grid = (
            (
                    torch.arange(-spline_order, grid_size + spline_order + 1) * h
                    + grid_range[0]
            )
            .expand(self.in_features, -1)
            .contiguous()
        )
        return grid

    def initialize_weights(self, out_features, in_features, grid_size, spline_order, scale_base, scale_spline,
                           enable_standalone_scale_spline):
        base_weight = torch.nn.Parameter(torch.Tensor(out_features, in_features))
        spline_weight = torch.nn.Parameter(
            torch.Tensor(out_features, in_features, grid_size + spline_order)
        )
        if enable_standalone_scale_spline:
            spline_scaler = torch.nn.Parameter(
                torch.Tensor(out_features, in_features)
            )
        else:
            spline_scaler = None
        torch.nn.init.kaiming_uniform_(base_weight, a=math.sqrt(5) * scale_base)
        torch.nn.init.kaiming_uniform_(spline_weight, a=math.sqrt(5) * scale_spline)
        if enable_standalone_scale_spline:
            torch.nn.init.kaiming_uniform_(spline_scaler, a=math.sqrt(5) * scale_spline)
        return base_weight, spline_weight, spline_scaler

    def b_splines(self, x: torch.Tensor):
        bases = self.calculate_b_spline_bases(x)
        return bases.contiguous()

    def calculate_b_spline_bases(self, x: torch.Tensor):
        grid: torch.Tensor = (
            self.grid
        )  # (in_features, grid_size + 2 * spline_order + 1)
        x = x.unsqueeze(-1)
        bases = ((x >= grid[:, :-1]) & (x < grid[:, 1:])).to(x.dtype)
        for k in range(1, self.spline_order + 1):
            bases = (
                            (x - grid[:, : -(k + 1)])
                            / (grid[:, k:-1] - grid[:, : -(k + 1)])
                            * bases[:, :, :-1]
                    ) + (
                            (grid[:, k + 1:] - x)
                            / (grid[:, k + 1:] - grid[:, 1:(-k)])
                            * bases[:, :, 1:]
                    )
        return bases

    def curve2coeff(self, x: torch.Tensor, y: torch.Tensor):
        A = self.b_splines(x).transpose(
            0, 1
        )  # (in_features, batch_size, grid_size + spline_order)
        B = y.transpose(0, 1)  # (in_features, batch_size, out_features)
        solution = torch.linalg.lstsq(
            A, B
        ).solution  # (in_features, grid_size + spline_order, out_features)
        result = solution.permute(
```

```python
            2, 0, 1
        )  # (out_features, in_features, grid_size + spline_order)
        return result.contiguous()

    @property
    def scaled_spline_weight(self):
        if self.enable_standalone_scale_spline:
            return self.spline_weight * self.spline_scaler.unsqueeze(-1)
        else:
            return self.spline_weight

    def forward(self, x: torch.Tensor):
        base_output = self.compute_base_output(x)
        spline_output = self.compute_spline_output(x)
        return base_output + spline_output

    def compute_base_output(self, x: torch.Tensor):
        return F.linear(self.base_activation(x), self.base_weight)

    def compute_spline_output(self, x: torch.Tensor):
        return F.linear(
            self.b_splines(x).view(x.size(0), -1),
            self.scaled_spline_weight.view(self.out_features, -1),
        )

    @torch.no_grad()
    def update_grid(self, x: torch.Tensor, margin=0.01):
        batch = x.size(0)

        splines = self.b_splines(x)  # (batch, in, coeff)
        splines = splines.permute(1, 0, 2)  # (in, batch, coeff)
        orig_coeff = self.scaled_spline_weight  # (out, in, coeff)
        orig_coeff = orig_coeff.permute(1, 2, 0)  # (in, coeff, out)
        unreduced_spline_output = torch.bmm(splines, orig_coeff)  # (in, batch, out)
        unreduced_spline_output = unreduced_spline_output.permute(
            1, 0, 2
        )  # (batch, in, out)

        x_sorted = torch.sort(x, dim=0)[0]
        grid_adaptive = x_sorted[
            torch.linspace(
                0, batch - 1, self.grid_size + 1, dtype=torch.int64, device=x.device
            )
        ]

        uniform_step = (x_sorted[-1] - x_sorted[0] + 2 * margin) / self.grid_size
        grid_uniform = (
                torch.arange(
                    self.grid_size + 1, dtype=torch.float32, device=x.device
                ).unsqueeze(1)
                * uniform_step
                + x_sorted[0]
                - margin
        )

        grid = self.grid_eps * grid_uniform + (1 - self.grid_eps) * grid_adaptive
        grid = torch.concatenate(
            [
                grid[:1]
                - uniform_step
                * torch.arange(self.spline_order, 0, -1, device=x.device).unsqueeze(1),
                grid,
                grid[-1:]
                + uniform_step
                * torch.arange(1, self.spline_order + 1, device=x.device).unsqueeze(1),
            ],
            dim=0,
        )

        self.grid.copy_(grid.T)
        self.spline_weight.data.copy_(self.curve2coeff(x, unreduced_spline_output))
```

```python
    def regularization_loss(self, regularize_activation=1.0, regularize_entropy=1.0):
        l1_fake = self.spline_weight.abs().mean(-1)
        regularization_loss_activation = l1_fake.sum()
        p = l1_fake / regularization_loss_activation
        regularization_loss_entropy = -torch.sum(p * p.log())
        return (
                regularize_activation * regularization_loss_activation
                + regularize_entropy * regularization_loss_entropy
        )


class KAN(torch.nn.Module):
    def __init__(
            self,
            layers_hidden,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KAN, self).__init__()
        self.grid_size = grid_size
        self.spline_order = spline_order

        # 構建 KAN 的層
        self.layers = self.build_layers(
            layers_hidden, grid_size, spline_order, scale_base, scale_spline, base_activation, grid_eps,
grid_range
        )

    def build_layers(self, layers_hidden, grid_size, spline_order, scale_base, scale_spline, base_activation,
grid_eps,
                     grid_range):
        layers = torch.nn.ModuleList()
        for in_features, out_features in zip(layers_hidden, layers_hidden[1:]):
            layers.append(
                KANLinear(
                    in_features,
                    out_features,
                    grid_size=grid_size,
                    spline_order=spline_order,
                    scale_base=scale_base,
                    scale_spline=scale_spline,
                    base_activation=base_activation,
                    grid_eps=grid_eps,
                    grid_range=grid_range,
                )
            )
        return layers

    def forward(self, x: torch.Tensor, update_grid=False):
        for layer in self.layers:
            if update_grid:
                layer.update_grid(x)
            x = layer(x)
        return x

    def regularization_loss(self, regularize_activation=1.0, regularize_entropy=1.0):
        return sum(
            layer.regularization_loss(regularize_activation, regularize_entropy)
            for layer in self.layers
        )
```

- 創建 Kolmogorov-Arnold Network 進行測試，資料集使用 MNIST 手寫數字辨識

```python
from EfficientKAN import KAN

# Train on MNIST
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from tqdm import tqdm

# Load MNIST
transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
)
trainset = torchvision.datasets.MNIST(
    root="./data", train=True, download=True, transform=transform
)
valset = torchvision.datasets.MNIST(
    root="./data", train=False, download=True, transform=transform
)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
valloader = DataLoader(valset, batch_size=64, shuffle=False)

# Define model
model = KAN([28 * 28, 64, 10])
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
# Define optimizer
optimizer = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-4)
# Define learning rate scheduler
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.8)

# Define loss
criterion = nn.CrossEntropyLoss()
for epoch in range(10):
    # Train
    model.train()
    with tqdm(trainloader) as pbar:
        for i, (images, labels) in enumerate(pbar):
            images = images.view(-1, 28 * 28).to(device)
            optimizer.zero_grad()
            output = model(images)
            loss = criterion(output, labels.to(device))
            loss.backward()
            optimizer.step()
            accuracy = (output.argmax(dim=1) == labels.to(device)).float().mean()
            pbar.set_postfix(loss=loss.item(), accuracy=accuracy.item(), lr=optimizer.param_groups[0]['lr'])

    # Validation
    model.eval()
    val_loss = 0
    val_accuracy = 0
    with torch.no_grad():
        for images, labels in valloader:
            images = images.view(-1, 28 * 28).to(device)
            output = model(images)
            val_loss += criterion(output, labels.to(device)).item()
            val_accuracy += (
                (output.argmax(dim=1) == labels.to(device)).float().mean().item()
            )
    val_loss /= len(valloader)
    val_accuracy /= len(valloader)

    # Update learning rate
    scheduler.step()

    print(
        f"Epoch {epoch + 1}, Val Loss: {val_loss}, Val Accuracy: {val_accuracy}"
    )
```

```python
# Print model weights
print("Trained Model Weights:")

for i, layer in enumerate(model.layers):
    print(f"Layer {i + 1}:")
    print("Spline Weights:")
    print(layer.spline_weight)
    print("Base Weights:")
    print(layer.base_weight)
    print()

# Save model weights (need to create KAN instance then "torch.load")
torch.save(model.state_dict(), "kan_mnist_weights.pth")
# Save the entire model (just get with "torch.load")
torch.save(model, "kan_mnist_model.pth")
```

- Model 於 MNIST 之訓練結果

```
100%|██████████| 938/938 [00:40<00:00, 23.07it/s, accuracy=1, loss=0.031, lr=0.000134]
Epoch 10, Val Loss: 0.08577567627701677, Val Accuracy: 0.9750199044585988
```

- 創建 Kolmogorov-Arnold Network 進行測試，函數擬合簡易乘法 a x b

```python
import torch
import torch.nn as nn
from tqdm import tqdm

from EfficientKAN import KAN


def test_mul():
    kan = KAN([2, 3, 3, 1], base_activation=nn.Identity)
    optimizer = torch.optim.LBFGS(kan.parameters(), lr=0.001)

    with tqdm(range(200)) as pbar:
        for i in pbar:
            loss, reg_loss = None, None

            def closure():
                optimizer.zero_grad()
                x = torch.rand(1024, 2)
                y = kan(x, update_grid=(i % 20 == 0))
                assert y.shape == (1024, 1)
                nonlocal loss, reg_loss
                u = x[:, 0]
                v = x[:, 1]
                loss = nn.functional.mse_loss(y.squeeze(-1), u * v)
                reg_loss = kan.regularization_loss(1, 0)
                (loss + 1e-5 * reg_loss).backward()
                return loss + reg_loss

            optimizer.step(closure)
            pbar.set_postfix(mse_loss=loss.item(), reg_loss=reg_loss.item())

    for layer in kan.layers:
        print(layer.spline_weight)

    torch.save(kan, 'model/kan_multiple_model.pth')
    torch.save(kan.state_dict(), "model/kan_multiple_weights.pth")

    # Test the trained model
    test_model(kan)


def test_model(model):
    model.eval()
    with torch.no_grad():
        test_x = torch.rand(1024, 2)
```

```python
        test_y = model(test_x)
        u = test_x[:, 0]
        v = test_x[:, 1]
        expected_y = u * v
        test_loss = nn.functional.mse_loss(test_y.squeeze(-1), expected_y)
        print(f"Test Loss: {test_loss.item():.4f}")


test_mul()
```

- Model 於函數擬合簡易乘法之訓練結果

```
100%|██████████| 200/200 [01:03<00:00,  3.13it/s, mse_loss=0.0938, reg_loss=9.12e-7]
```

訓練模型 Quantization 處理

> **Todo**
>
> 為了解決模型 Floating Point 將會對電路設計產生的影響，需要先針對模型權重的 Quantization 進行實驗，用以尋找合適的 Integer Quantization 模式。

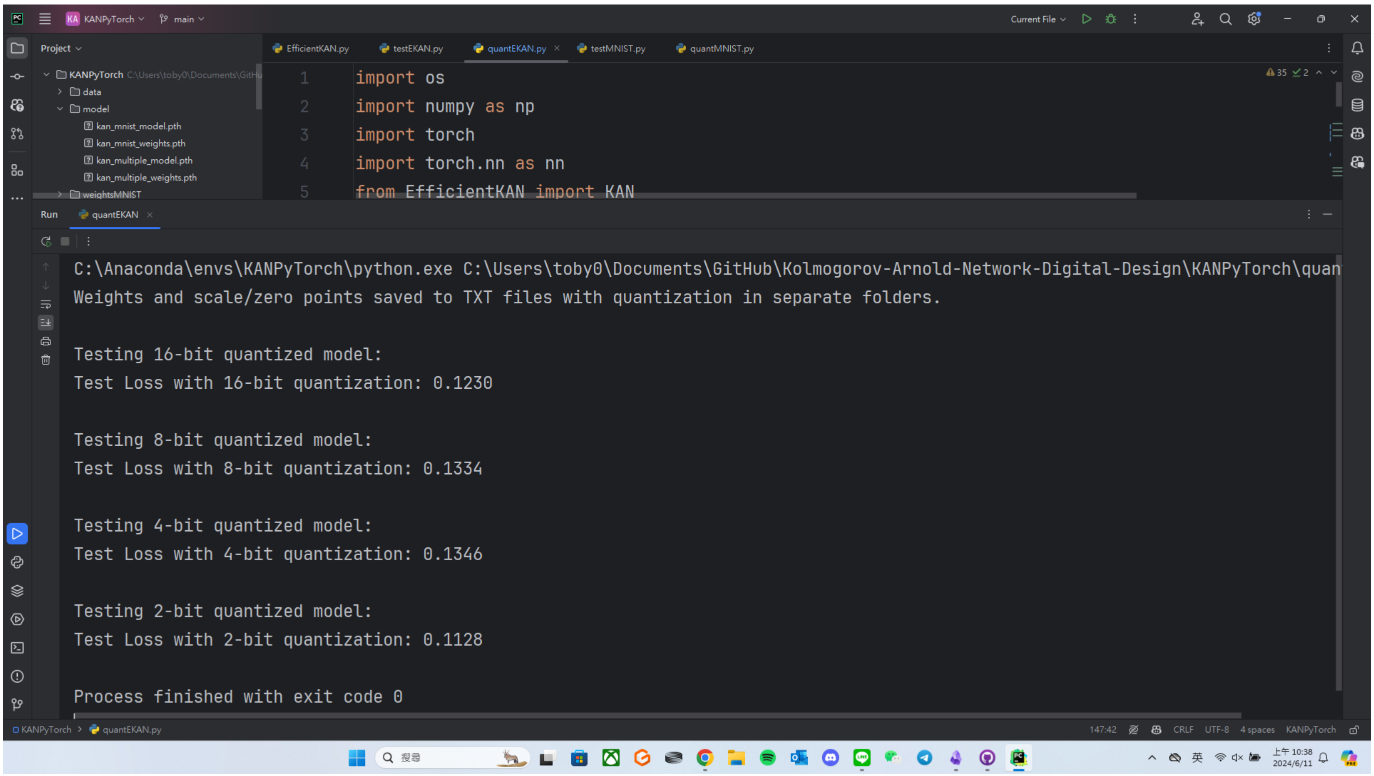
> **Check**
>
> Multiplication 測試模型 Integer Quantization 比較，INT16、INT8、INT4、INT2。

- Quantization 功能設計、新權重保存和新權重評估（更改為二進制權重輸出以及單獨處理 Scale 和 Zero Point）

```python
import os
import numpy as np
import torch
import torch.nn as nn
from EfficientKAN import KAN

# 加載已訓練的模型
model_path = 'model/kan_multiple_weights.pth'
model = KAN([2, 3, 3, 1], base_activation=nn.Identity)
model.load_state_dict(torch.load(model_path))

# 量化工具
def quantize_tensor(tensor, num_bits):
    qmin = 0.
    qmax = 2. ** num_bits - 1.

    min_val, max_val = tensor.min(), tensor.max()
    scale = (max_val - min_val) / (qmax - qmin)
    initial_zero_point = qmin - min_val / scale

    zero_point = 0
    if initial_zero_point < qmin:
        zero_point = qmin
    elif initial_zero_point > qmax:
        zero_point = qmax
    else:
        zero_point = initial_zero_point

    zero_point = int(zero_point)
    quantized_tensor = (tensor / scale + zero_point).round().clamp(qmin, qmax)
    quantized_tensor = quantized_tensor.int()

    return quantized_tensor, scale, zero_point

# 將浮點數轉換為二進制字符串
def float_to_binary(value):
    import struct
    [d] = struct.unpack(">Q", struct.pack(">d", value))
```

```python
    return f'{d:064b}'

# 將整數轉換為二進制字符串
def int_to_binary(value, num_bits):
    return f'{value:0{num_bits}b}'

# 將權重保存到單獨的 TXT 檔案
def save_layer_weights_to_txt(model, layer_index, base_dir, num_bits):
    layer = model.layers[layer_index]
    folder_name = os.path.join(base_dir, f"{num_bits}bits")
    os.makedirs(folder_name, exist_ok=True)

    base_weight_file = os.path.join(folder_name, f"layer{layer_index}_base_weight.txt")
    spline_weight_file = os.path.join(folder_name, f"layer{layer_index}_spline_weight.txt")
    spline_scaler_file = os.path.join(folder_name, f"layer{layer_index}_spline_scaler.txt")

    base_weight_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_base_weight.txt")
    base_weight_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_base_weight.txt")

    spline_weight_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_spline_weight.txt")
    spline_weight_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_spline_weight.txt")

    spline_scaler_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_spline_scaler.txt")
    spline_scaler_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_spline_scaler.txt")

    with open(base_weight_file, 'w') as f:
        base_weight_data = layer.base_weight.detach().cpu().numpy()
        quantized_base_weight, scale, zero_point = quantize_tensor(torch.tensor(base_weight_data), num_bits)
        for value in quantized_base_weight.flatten():
            f.write(f'{int_to_binary(value, num_bits)}\n')
    with open(base_weight_scale_file, 'w') as f:
        f.write(f"{scale}\n")
    with open(base_weight_zero_point_file, 'w') as f:
        f.write(f"{zero_point}\n")

    with open(spline_weight_file, 'w') as f:
        spline_weight_data = layer.spline_weight.detach().cpu().numpy()
        quantized_spline_weight, scale, zero_point = quantize_tensor(torch.tensor(spline_weight_data), num_bits)
        for value in quantized_spline_weight.flatten():
            f.write(f'{int_to_binary(value, num_bits)}\n')
    with open(spline_weight_scale_file, 'w') as f:
        f.write(f"{scale}\n")
    with open(spline_weight_zero_point_file, 'w') as f:
        f.write(f"{zero_point}\n")

    with open(spline_scaler_file, 'w') as f:
        spline_scaler_data = layer.spline_scaler.detach().cpu().numpy()
        quantized_spline_scaler, scale, zero_point = quantize_tensor(torch.tensor(spline_scaler_data), num_bits)
        for value in quantized_spline_scaler.flatten():
            f.write(f'{int_to_binary(value, num_bits)}\n')
    with open(spline_scaler_scale_file, 'w') as f:
        f.write(f"{scale}\n")
    with open(spline_scaler_zero_point_file, 'w') as f:
        f.write(f"{zero_point}\n")

# 從 TXT 文件加載量化後的權重
def binary_to_float(binary_str):
    import struct
    bf = int(binary_str, 2)
    return struct.unpack(">d", struct.pack(">Q", bf))[0]

def binary_to_int(binary_str):
    return int(binary_str, 2)

def read_quantized_file(file_path, num_bits):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    quantized_values = np.array([binary_to_int(v.strip()) for v in lines])
    return quantized_values

def read_scale_zero_point(file_path, is_float):
```

```python
    with open(file_path, 'r') as f:
        value = f.readline().strip()
    return float(value) if is_float else int(value)

def load_quantized_weights_from_txt(model, layer_index, base_dir, num_bits):
    layer = model.layers[layer_index]
    folder_name = os.path.join(base_dir, f"{num_bits}bits")

    base_weight_file = os.path.join(folder_name, f"layer{layer_index}_base_weight.txt")
    spline_weight_file = os.path.join(folder_name, f"layer{layer_index}_spline_weight.txt")
    spline_scaler_file = os.path.join(folder_name, f"layer{layer_index}_spline_scaler.txt")

    base_weight_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_base_weight.txt")
    base_weight_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_base_weight.txt")

    spline_weight_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_spline_weight.txt")
    spline_weight_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_spline_weight.txt")

    spline_scaler_scale_file = os.path.join(folder_name, f"layer{layer_index}_scale_spline_scaler.txt")
    spline_scaler_zero_point_file = os.path.join(folder_name, f"layer{layer_index}_zero_point_spline_scaler.txt")

    quantized_base_weight = read_quantized_file(base_weight_file, num_bits)
    base_weight_scale = read_scale_zero_point(base_weight_scale_file, is_float=True)
    base_weight_zero_point = read_scale_zero_point(base_weight_zero_point_file, is_float=False)
    layer.base_weight.data = torch.tensor((quantized_base_weight - base_weight_zero_point) * base_weight_scale,
                                          dtype=torch.float32).view_as(layer.base_weight)

    quantized_spline_weight = read_quantized_file(spline_weight_file, num_bits)
    spline_weight_scale = read_scale_zero_point(spline_weight_scale_file, is_float=True)
    spline_weight_zero_point = read_scale_zero_point(spline_weight_zero_point_file, is_float=False)
    layer.spline_weight.data = torch.tensor((quantized_spline_weight - spline_weight_zero_point) *
spline_weight_scale,
                                            dtype=torch.float32).view_as(layer.spline_weight)

    if os.path.exists(spline_scaler_file):
        quantized_spline_scaler = read_quantized_file(spline_scaler_file, num_bits)
        spline_scaler_scale = read_scale_zero_point(spline_scaler_scale_file, is_float=True)
        spline_scaler_zero_point = read_scale_zero_point(spline_scaler_zero_point_file, is_float=False)
        layer.spline_scaler.data = torch.tensor(
            (quantized_spline_scaler - spline_scaler_zero_point) * spline_scaler_scale,
dtype=torch.float32).view_as(
            layer.spline_scaler)

# 測試量化後的模型性能
def test_quantized_model(model, base_dir, num_bits):
    for i in range(len(model.layers)):
        load_quantized_weights_from_txt(model, i, base_dir, num_bits)

    model.eval()
    with torch.no_grad():
        test_x = torch.rand(1024, 2)
        test_y = model(test_x)
        u = test_x[:, 0]
        v = test_x[:, 1]
        expected_y = u * v
        test_loss = nn.functional.mse_loss(test_y.squeeze(-1), expected_y)
        print(f"Test Loss with {num_bits}-bit quantization: {test_loss.item():.4f}")

# 呼叫函數保存每層的權重，使用16位、8位和4位量化和其他位數
bit_levels = [16, 8, 4, 2]
base_dir = 'weightsMultiplication'
for num_bits in bit_levels:
    for i in range(len(model.layers)):
        save_layer_weights_to_txt(model, i, base_dir, num_bits)

print("Weights and scale/zero points saved to TXT files with quantization in separate folders.")

# 測試不同量化位數的模型
for num_bits in bit_levels:
    print(f"\nTesting {num_bits}-bit quantized model:")
    test_quantized_model(model, base_dir, num_bits)
```



✅ Check

MNIST 測試模型 Integer Quantization 比較，INT16、INT8、INT4、INT2。

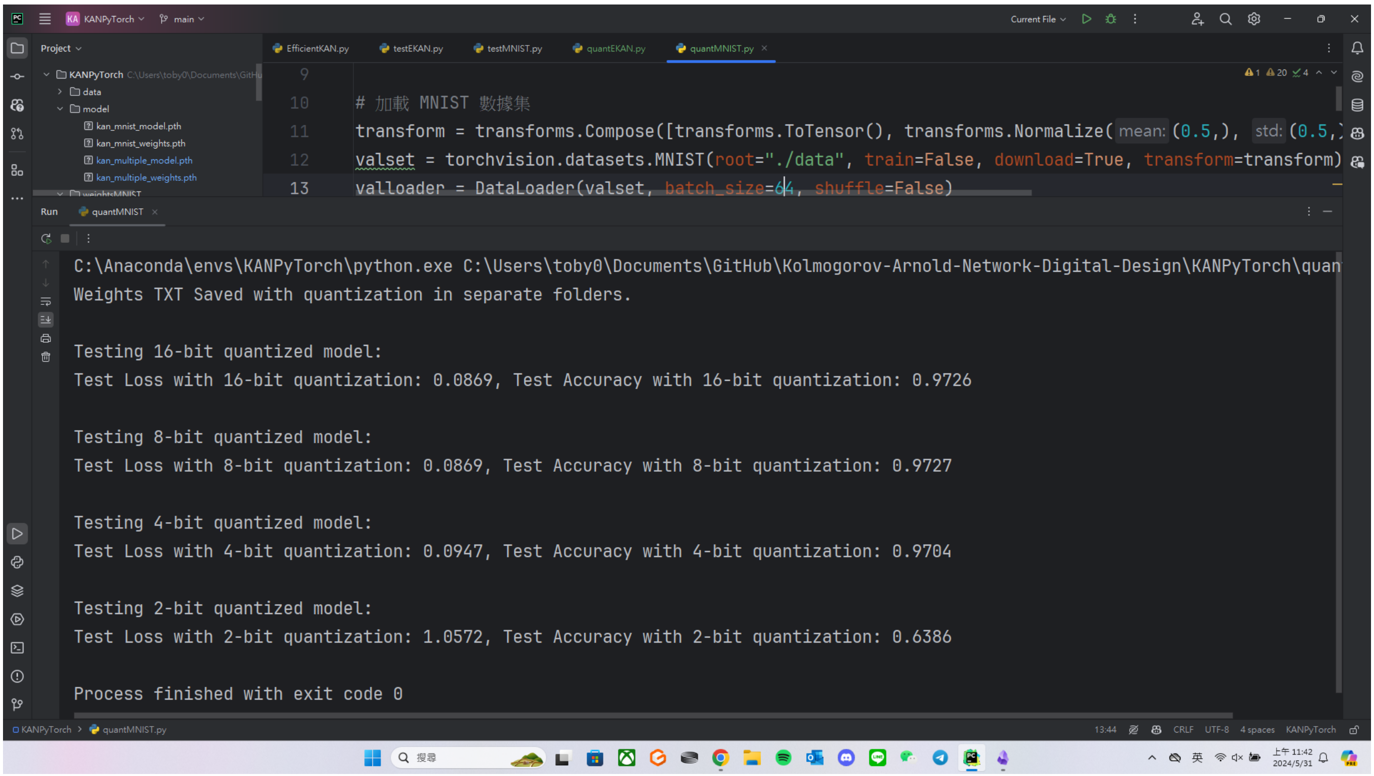
- Quantization 功能設計、新權重保存和新權重評估

```
import os

import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# 加載 MNIST 數據集
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
valset = torchvision.datasets.MNIST(root="./data", train=False, download=True, transform=transform)
valloader = DataLoader(valset, batch_size=64, shuffle=False)

# 加載已訓練的模型
model_path = 'model/kan_mnist_model.pth'
model = torch.load(model_path)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# 定義損失函數
criterion = nn.CrossEntropyLoss()


# 量化工具
def quantize_tensor(tensor, num_bits):
    qmin = 0.
    qmax = 2. ** num_bits - 1.

    min_val, max_val = tensor.min(), tensor.max()
    scale = (max_val - min_val) / (qmax - qmin)
    initial_zero_point = qmin - min_val / scale

    zero_point = 0
    if initial_zero_point < qmin:
```

```python
        zero_point = qmin
    elif initial_zero_point > qmax:
        zero_point = qmax
    else:
        zero_point = initial_zero_point

    zero_point = int(zero_point)
    quantized_tensor = (tensor / scale + zero_point).round().clamp(qmin, qmax)
    quantized_tensor = quantized_tensor.int()

    return quantized_tensor, scale, zero_point


# 將權重保存到單獨的 TXT 檔案
def save_layer_weights_to_txt(model, layer_index, base_dir, num_bits):
    layer = model.layers[layer_index]
    folder_name = os.path.join(base_dir, f"{num_bits}bits")
    os.makedirs(folder_name, exist_ok=True)

    base_weight_file = os.path.join(folder_name, f"layer{layer_index}_base_weight.txt")
    spline_weight_file = os.path.join(folder_name, f"layer{layer_index}_spline_weight.txt")
    spline_scaler_file = os.path.join(folder_name, f"layer{layer_index}_spline_scaler.txt")

    with open(base_weight_file, 'w') as f:
        base_weight_data = layer.base_weight.detach().cpu().numpy()
        quantized_base_weight, scale, zero_point = quantize_tensor(torch.tensor(base_weight_data), num_bits)
        f.write(f"scale: {scale}\n")
        f.write(f"zero_point: {zero_point}\n")
        for value in quantized_base_weight.flatten():
            f.write(f'{value}\n')

    with open(spline_weight_file, 'w') as f:
        spline_weight_data = layer.spline_weight.detach().cpu().numpy()
        quantized_spline_weight, scale, zero_point = quantize_tensor(torch.tensor(spline_weight_data), num_bits)
        f.write(f"scale: {scale}\n")
        f.write(f"zero_point: {zero_point}\n")
        for value in quantized_spline_weight.flatten():
            f.write(f'{value}\n')

    if layer.spline_scaler is not None:
        with open(spline_scaler_file, 'w') as f:
            spline_scaler_data = layer.spline_scaler.detach().cpu().numpy()
            quantized_spline_scaler, scale, zero_point = quantize_tensor(torch.tensor(spline_scaler_data),
num_bits)
            f.write(f"scale: {scale}\n")
            f.write(f"zero_point: {zero_point}\n")
            for value in quantized_spline_scaler.flatten():
                f.write(f'{value}\n')


# 呼叫函數保存每層的權重，使用16位、8位和4位量化
bit_levels = [16, 8, 4, 2]
base_dir = 'weightsMNIST'
for num_bits in bit_levels:
    for i in range(len(model.layers)):
        save_layer_weights_to_txt(model, i, base_dir, num_bits)

print("Weights TXT Saved with quantization in separate folders.")


# 從 TXT 文件加載量化後的權重
def load_quantized_weights_from_txt(model, layer_index, base_dir, num_bits):
    layer = model.layers[layer_index]
    folder_name = os.path.join(base_dir, f"{num_bits}bits")

    base_weight_file = os.path.join(folder_name, f"layer{layer_index}_base_weight.txt")
    spline_weight_file = os.path.join(folder_name, f"layer{layer_index}_spline_weight.txt")
    spline_scaler_file = os.path.join(folder_name, f"layer{layer_index}_spline_scaler.txt")

    def read_quantized_file(file_path):
        with open(file_path, 'r') as f:
```

```
            lines = f.readlines()
        scale = float(lines[0].strip().split(": ")[1])
        zero_point = int(lines[1].strip().split(": ")[1])
        quantized_values = np.array([int(v.strip()) for v in lines[2:]])
        return quantized_values, scale, zero_point

    quantized_base_weight, scale, zero_point = read_quantized_file(base_weight_file)
    layer.base_weight.data = torch.tensor((quantized_base_weight - zero_point) * scale,
dtype=torch.float32).view_as(
        layer.base_weight)

    quantized_spline_weight, scale, zero_point = read_quantized_file(spline_weight_file)
    layer.spline_weight.data = torch.tensor((quantized_spline_weight - zero_point) * scale,
                                            dtype=torch.float32).view_as(layer.spline_weight)

    if os.path.exists(spline_scaler_file):
        quantized_spline_scaler, scale, zero_point = read_quantized_file(spline_scaler_file)
        layer.spline_scaler.data = torch.tensor((quantized_spline_scaler - zero_point) * scale,
                                                dtype=torch.float32).view_as(layer.spline_scaler)


# 測試量化後的模型性能
def test_quantized_model(model, base_dir, num_bits):
    for i in range(len(model.layers)):
        load_quantized_weights_from_txt(model, i, base_dir, num_bits)

    model.eval()
    val_loss = 0
    val_accuracy = 0
    with torch.no_grad():
        for images, labels in valloader:
            images = images.view(-1, 28 * 28).to(device)
            output = model(images)
            val_loss += criterion(output, labels.to(device)).item()
            val_accuracy += ((output.argmax(dim=1) == labels.to(device)).float().mean().item())
    val_loss /= len(valloader)
    val_accuracy /= len(valloader)
    print(
        f"Test Loss with {num_bits}-bit quantization: {val_loss:.4f}, Test Accuracy with {num_bits}-bit
quantization: {val_accuracy:.4f}")


# 測試不同量化位數的模型
for num_bits in bit_levels:
    print(f"\nTesting {num_bits}-bit quantized model:")
    test_quantized_model(model, base_dir, num_bits)
```

- 評估結果 (除 2-bit Quantization 以外，其餘正確率接可維持在 97%，與 Quantization 前的結果相差無幾)



方法論合成 Grafcet 控制器電路

Success

更換測試資料及 Network 設計以求可以將電路設計容納進去（學習函數擬合簡易乘法，KAN 的架構為 [2, 3, 3, 1]）。

- KANLayer 實現（KANLayer.v）

```
module KANLinear #(parameter IN_FEATURES = 2, OUT_FEATURES = 3, integer LAYER_NUM = 0) (
    input clk,
    input reset,
    input [15:0] data_in [IN_FEATURES-1:0],
    output reg [15:0] data_out [OUT_FEATURES-1:0]
);

    reg [15:0] base_weight [OUT_FEATURES*IN_FEATURES-1:0];
    reg [15:0] spline_weight [OUT_FEATURES*IN_FEATURES-1:0];
    reg [15:0] base_weight_2d [OUT_FEATURES-1:0][IN_FEATURES-1:0];
    reg [15:0] spline_weight_2d [OUT_FEATURES-1:0][IN_FEATURES-1:0];

    integer i, j;

    initial begin
        if (LAYER_NUM == 0) begin
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_0.txt", base_weight);
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_0.txt", spline_weight);
        end else if (LAYER_NUM == 1) begin
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_1.txt", base_weight);
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_1.txt", spline_weight);
        end else if (LAYER_NUM == 2) begin
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_2.txt", base_weight);
            $readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_2.txt", spline_weight);
        end

        for (i = 0; i < OUT_FEATURES; i = i + 1) begin
            for (j = 0; j < IN_FEATURES; j = j + 1) begin
                base_weight_2d[i][j] = base_weight[i * IN_FEATURES + j];
                spline_weight_2d[i][j] = spline_weight[i * IN_FEATURES + j];
            end
```

```verilog
        end
    end

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                data_out[i] <= 16'd0;
            end
        end else begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                data_out[i] <= 16'd0;
                for (j = 0; j < IN_FEATURES; j = j + 1) begin
                    data_out[i] <= data_out[i] + base_weight_2d[i][j] * data_in[j];
                end
                if (data_out[i] < 16'd0) begin
                    data_out[i] <= 16'd0;
                end
            end
        end
    end

endmodule
```

> **⚠ Caution**
>
> MNIST 版本之 Reference

- KANLayer 實現 (KANLayer.v)

```verilog
module KANLayer #(
    parameter IN_FEATURES = 784,
    parameter OUT_FEATURES = 64,
    parameter SCALE = 256,  // Quantization scale factor
    parameter BASE_WEIGHT_FILE = "C:/intelFPGA_lite/18.1/KAN/base_weight_layer_0.txt",
    parameter SPLINE_WEIGHT_FILE = "C:/intelFPGA_lite/18.1/KAN/spline_weight_layer_0.txt"
)(
    input wire clk,
    input wire reset,
    input wire [7:0] in_data [0:IN_FEATURES-1],  // Input data
    output reg [7:0] out_data [0:OUT_FEATURES-1] // Output data
);
    // Weights stored in on-chip memory (BRAM)
    reg signed [15:0] base_weights [0:OUT_FEATURES*IN_FEATURES-1];
    reg signed [15:0] spline_weights [0:OUT_FEATURES*IN_FEATURES-1];

    // Load weights from memory (initialization)
    initial begin
        $readmemh(BASE_WEIGHT_FILE, base_weights);
        $readmemh(SPLINE_WEIGHT_FILE, spline_weights);
    end

    // Output registers
    reg signed [31:0] base_output [0:OUT_FEATURES-1];
    reg signed [31:0] spline_output [0:OUT_FEATURES-1];
    reg signed [31:0] total_output [0:OUT_FEATURES-1];

    integer i, j;

    // Forward pass
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                base_output[i] <= 0;
                spline_output[i] <= 0;
                total_output[i] <= 0;
            end
        end else begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                base_output[i] <= 0;
```

```
                spline_output[i] <= 0;
                for (j = 0; j < IN_FEATURES; j = j + 1) begin
                    base_output[i] <= base_output[i] + in_data[j] * base_weights[i*IN_FEATURES + j];
                    spline_output[i] <= spline_output[i] + in_data[j] * spline_weights[i*IN_FEATURES + j];
                end
                total_output[i] <= (base_output[i] + spline_output[i]) / SCALE;  // Combine and scale the outputs
                out_data[i] <= total_output[i][15:8];  // Convert to 8-bit output
            end
        end
    end
endmodule
```

ChatGPT 合成 Grafcet Datapath 電路

> **Success**
>
> 更換測試資料及 Network 設計以求可以將電路設計容納進去（學習函數擬合簡易乘法，KAN 的架構為 [2, 3, 3, 1]）。

- 完整 Network 實現（KAN.v）

```
module KAN #(parameter IN_FEATURES = 2, L1_FEATURES = 3, L2_FEATURES = 3, OUT_FEATURES = 1) (
    input clk,
    input reset,
    input [15:0] input_data [IN_FEATURES-1:0],
    output [15:0] output_data
);

    wire [15:0] layer1_out [L1_FEATURES-1:0];
    wire [15:0] layer2_out [L2_FEATURES-1:0];
    wire [15:0] layer3_out [OUT_FEATURES-1:0];

    KANLinear #(.IN_FEATURES(IN_FEATURES), .OUT_FEATURES(L1_FEATURES), .LAYER_NUM(0)) layer1 (
        .clk(clk),
        .reset(reset),
        .data_in(input_data),
        .data_out(layer1_out)
    );

    KANLinear #(.IN_FEATURES(L1_FEATURES), .OUT_FEATURES(L2_FEATURES), .LAYER_NUM(1)) layer2 (
        .clk(clk),
        .reset(reset),
        .data_in(layer1_out),
        .data_out(layer2_out)
    );

    KANLinear #(.IN_FEATURES(L2_FEATURES), .OUT_FEATURES(OUT_FEATURES), .LAYER_NUM(2)) layer3 (
        .clk(clk),
        .reset(reset),
        .data_in(layer2_out),
        .data_out(layer3_out)
    );

    assign output_data = layer3_out[0];

endmodule
```

> **Caution**
>
> MNIST 版本之 Reference

- 完整 Network 實現（KAN.v）

```
module KAN (
    input wire clk,
```

```
    input wire reset,
    input wire [7:0] in_data [0:783],  // 28x28 = 784 pixels, 8-bit each
    output wire [7:0] out_data [0:9]   // 10 classes, 8-bit each
);
    // Internal signals for each layer
    wire [7:0] layer1_out [0:63];
    wire [7:0] layer2_out [0:9];

    // Instantiate layers
    KANLayer #(
        .IN_FEATURES(784),
        .OUT_FEATURES(64),
        .BASE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/base_weight_layer_0.txt"),
        .SPLINE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/spline_weight_layer_0.txt")
    ) layer1 (
        .clk(clk),
        .reset(reset),
        .in_data(in_data),
        .out_data(layer1_out)
    );

    KANLayer #(
        .IN_FEATURES(64),
        .OUT_FEATURES(10),
        .BASE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/base_weight_layer_1.txt"),
        .SPLINE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/spline_weight_layer_1.txt")
    ) layer2 (
        .clk(clk),
        .reset(reset),
        .in_data(layer1_out),
        .out_data(layer2_out)
    );

    // Connect the final output
    assign out_data = layer2_out;
endmodule
```

FPGA 整合驗證

✅ Success

更換測試資料及 Network 設計以求可以將電路設計容納進去（學習函數擬合簡易乘法，KAN 的架構為 [2, 3, 3, 1]）。

- Testbench 實現（Testbench.v）

```
module TestBench;
    reg clk;
    reg reset;
    reg [15:0] input_data [0:1];
    wire [15:0] output_data;

    KAN #(.IN_FEATURES(2), .L1_FEATURES(3), .L2_FEATURES(3), .OUT_FEATURES(1)) kan (
        .clk(clk),
        .reset(reset),
        .input_data(input_data),
        .output_data(output_data)
    );

    initial begin
        clk = 0;
        reset = 1;
        input_data[0] = 16'd0;
        input_data[1] = 16'd0;
        #10 reset = 0;

        // test data 1
        input_data[0] = 16'd50;
```

```
        input_data[1] = 16'd30;
        #10;
        $display("Output (Test 1): %d", output_data);

        // test data 2
        input_data[0] = 16'd100;
        input_data[1] = 16'd200;
        #10;
        $display("Output (Test 2): %d", output_data);

        // test data 3
        input_data[0] = 16'd150;
        input_data[1] = 16'd250;
        #10;
        $display("Output (Test 3): %d", output_data);

        // test data 4
        input_data[0] = 16'd75;
        input_data[1] = 16'd125;
        #10;
        $display("Output (Test 4): %d", output_data);

        // test data 5
        input_data[0] = 16'd175;
        input_data[1] = 16'd225;
        #10;
        $display("Output (Test 5): %d", output_data);
    end

    always #5 clk = ~clk;

endmodule
```

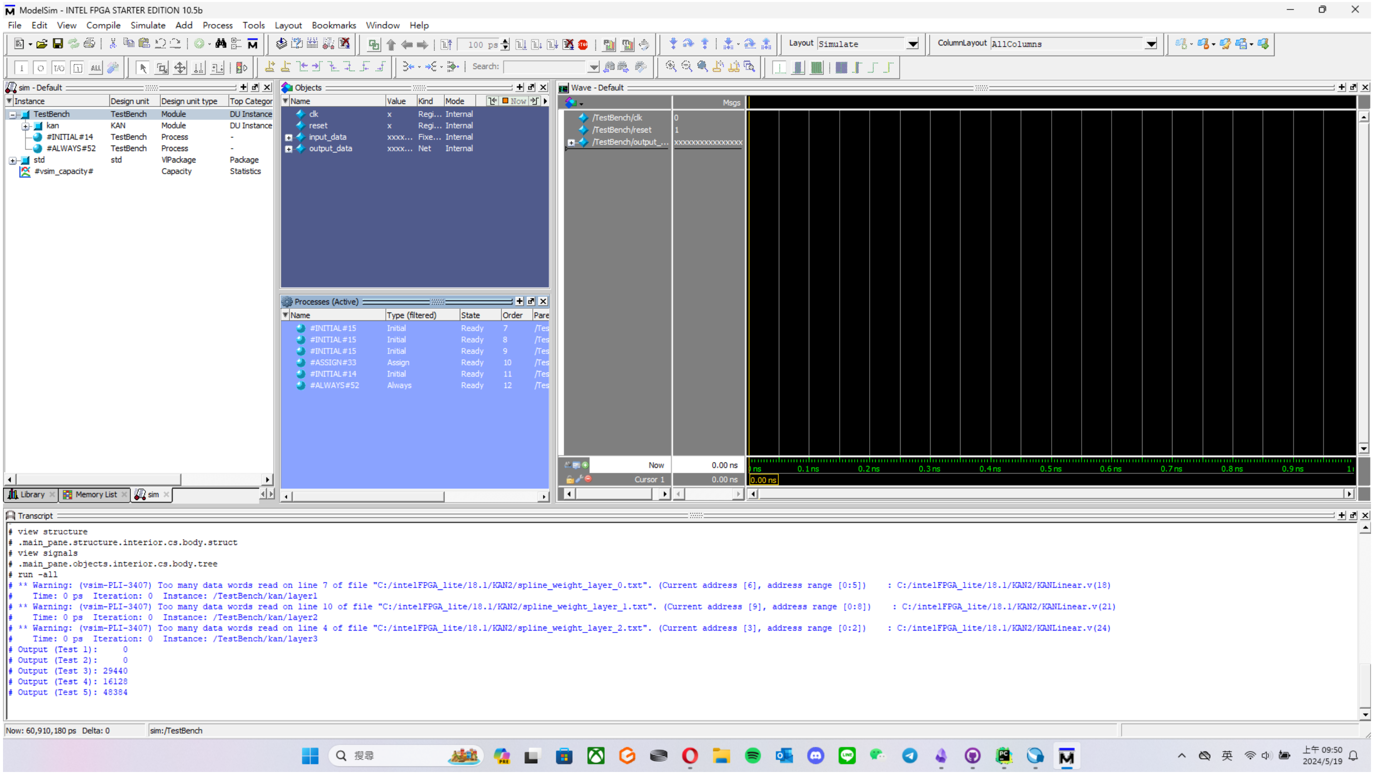
Warning

硬體設計完仿真的結果如下，誤差老實說非常大，基本上是 Quantization 後的 Weight 發生狀況。

Todo

目前主要的問題在於原生的 Verilog 不支持浮點運算，而這剛好是 KAN 最大的痛點，畢竟 KAN 與 MLP 最大的不同就在於是使用曲線去做合成，基本上出來的權重都會是小數。就算完成 Quantization，也只是 Weight 本身的值是 INT，依舊會需要處理浮點數的 Scaler 才能進行正確的權重還原，也因為可以進行 Weight 還原，使得 PyTorch 在 Quantization 後依舊可以有好結果（畢竟 Python 是支援浮點運算的）。目前還沒有想到甚麼比較好的解決方法，只是單純去查了一下說 HLS 可以支持單精度浮點和雙精度浮點運算，那之後可能會朝相關方向前進，暫時就不調整目前 Verilog 的結果。

- ModelSim 仿真結果



Pipeline 架構處理