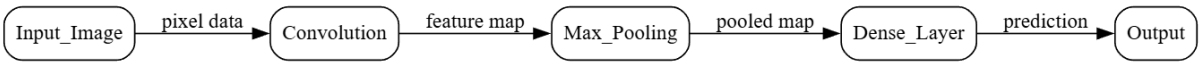
CNN HARDWARE ACCELERATOR

Simple Fpga based cnn accelerator for image classification (cat and dog)

Data set used :- **CIFAR 10** (3 & 5)

Block Diagram



Module	Input Size	Output Size
cnn_top	28×28 image (784 pixels)	Prediction (1 bit)
control_fsm	Control signals	Control signals
processing_unit	8-bit pixel and weight	16-bit MAC result
max_pooling_unit	16-bit feature map pixels (26×26)	13×13 pooled map pixels
dense_layer	169×16-bit features	32-bit class scores

Weight and Input

kernel is given to the model as **kernel.hex** in the **cnn_top.v**

Weights of cat and dog from CIFAR 10 is given as **weights_c0.hex** and **weights_c1.hex** in **dense_layer.v**

Module Descriptions

`Cnnn_top.v`:-The master controller that sequences CNN stages (Conv → Pool → Dense → Predict).

`control_fsm.v :-`Controls convolutional scanning across the input image with a 3×3 kernel, managing pixel and weight addressing.

`processing_unit.v :-`Performs Multiply-Accumulate (MAC) operations for each 3×3 convolution kernel.

`max_pooling_unit.v :-`Applies 2×2 max pooling on the resulting 26×26 feature map to reduce it to 13×13

`dense_layer.v :-`Implements the final fully connected layer with two output neurons for classification.

`Cat_or_dog.v :-` Test bench that used to test the proper working of cnn accelerator.after the entire processing test bench will print the predicted output and print it in tcl console

Data Flow and FSM Explanation

- The master FSM in `cnnn_top.v` transitions through:
  `IDLE → CONV_START → CONV_WAIT → POOL_START → POOL_WAIT → DENSE_START → DENSE_FEED → DENSE_WAIT → PREDICT → DONE_STATE`.
- Each subsystem (Conv, Pool, Dense) raises a `done` flag when finished, triggering the next stage.
- Data passes through sequential buffers: pixel data → feature map → pooled map → dense feature stream.
- The process is fully serialized at the stage level:

How the process actually works in your project:

1. Input pixels (784 pixels, 28×28 image)
   The entire image is fed into the processing unit (convolution module) *sequentially* (pixel by pixel), not all at once, controlled by the FSM.
2. Convolution layer ("Multiply with Kernel, ReLU applied")
   - The processing unit multiplies small patches (3×3 windows, 9 pixels) with fixed kernel weights.
   - The multiplication results are summed (MAC operation) producing a feature map.
   - This process repeats for different regions across the image (sliding window).
   - The activation (ReLU) is applied to the resulting feature map values.
3. Feature map (26×26 pixels)
   After convolution, you get a feature map of size roughly 26×26. This is not the original 784 pixels but the result of the convolution operations.
4. Pooling layer (2×2 max pooling)

    - The 26×26 feature map is downsampled by taking the maximum in each 2×2 window.
    - This reduces the size to approximately 13×13 pixels (actually 13×13 = 169 pixels).
    - Note: This is a reduction from the feature map, not directly from the raw input pixels.
5. Flatten and Fully Connected (Dense) layer
    - The 13×13 pooled feature map is flattened to 169 features.
    - These features are fed into the dense layer, which computes two class scores (e.g., cat/dog).
6. Decision (Prediction)
    - The class with the higher score is selected as the prediction.

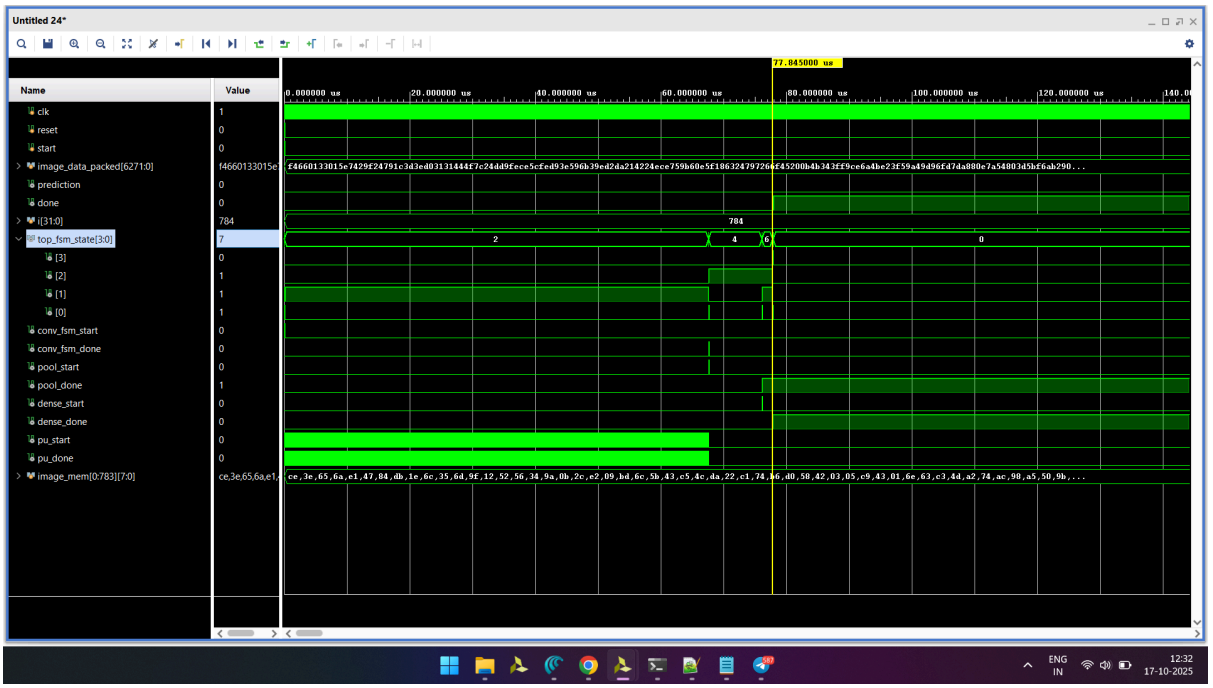
Testbench Features(`cat_or_dog.v`)

- Loads a 28×28 grayscale image into the CNN input vector.
- Provides reset, start, and monitors prediction and done signals for correct operation.
- Connects to key internal FSM and control signals for waveform debugging (FSM states, start/done signals of all stages).
- Prints the classification result to the simulation console.

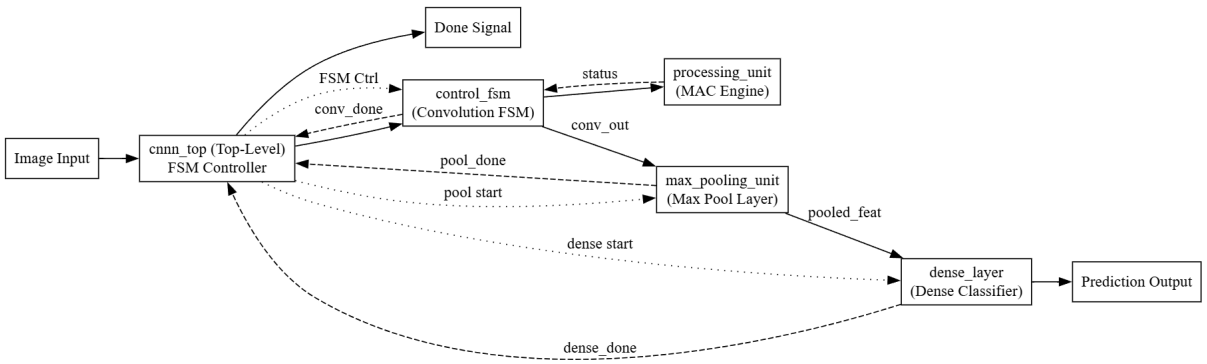
Key Observations

- The main FSM (`top_fsm_state[3:0]`) transitions correctly through its states, ending at state 7 (likely the `DONE` state), confirming that all pipeline stages executed fully.
- The `done` signal goes high at the end of the process, indicating the accelerator finished inference as expected.
- Control signals such as `conv_fsm_start/done`, `pool_start/done`, `dense_start/done`, and `pu_start/done` each activate and de-assert in proper sequence, confirming that convolution, pooling, and dense stages are chained correctly and each module completes its task in coordination with the top-level FSM.
- The `prediction` output is driven at the end, signifying that the device is issuing a valid classification result for the test image.

Simulation output



FLOW



Summary of What This Project Is

This Verilog design represents a small CNN for FPGA-based inference on grayscale images—roughly analogous to the first few layers of CIFRA 10 image classification. It's a clean educational architecture demonstrating:

- FSM-based control of neural network computations,

- Convolution, pooling, and dense layers implemented as standalone units,
- Integration of multiple levels of state machines and memory-based feature passing.
- The project uses fixed-width signed integers with **truncation** as needed, which can be considered a basic form of fixed-point integer arithmetic without fractional handling.

Conclusion

Due to the low accuracy (<60%) of the trained model, the predicted output may show errors. This limitation is acknowledged, and future work will focus on improving model accuracy to enhance the reliability of the CNN accelerator's predictions.