Multi-Play Multi-Armed Bandit Algorithms for Wideband Sensing: K+-SL

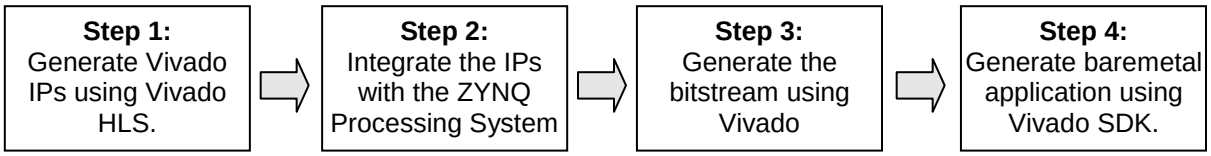
Introduction

In this lab, we will use Vivado High Level Synthesis (HLS), Vivado IPI and Software Development Kit (SDK) to create a peripheral capable of executing the K+-SL algorithm using ARM Cortex-A9 processor system on ZYNQ. We will use ZC706 to verify the design in hardware using the JTAG to configure the FPGA.

GitHub Link: Sai-Santosh-99/MPMAB-HW/K+SL

Note: The subset-learning algorithm is now K+-SL algorithm.

General Flow



Algorithm Description

The fundamental steps of the K+-SL algorithm for a setting with number of arms, N=4 and minimum subset size, K=2, along with hardware-software partitioning are discussed below:

Step 1: For the first 11 (no. of possible subsets) time slots, select all the possible subsets sequentially. Otherwise, select the subset as determined by step V. This is done inside the processor as the whole environment for the MPMAB problem is modelled inside the processor.

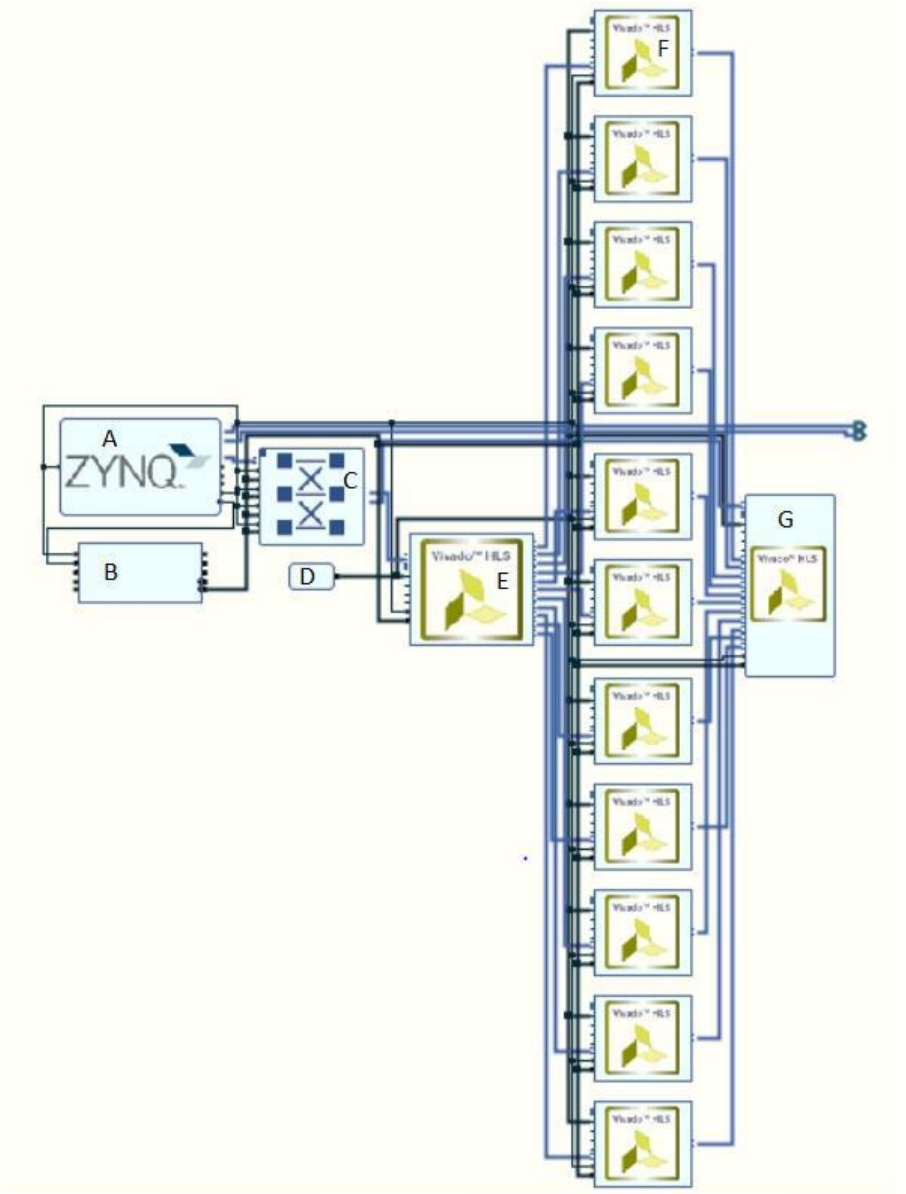
Step 2: Play the selected subset and determine whether among the arms of the selected subset, the number of 'occupied arms' is more than K = 2. This is also done inside the processor.

Step 3: If the number of 'occupied arms' inside the subset is more than K = 2, then add a reward of 0 to the X values of all subsets. Else, the appropriate reward is added to the X values of the selected subset. The T value of the selected subset is incremented by 1 irrespective of the condition outcome of the previous step. This is done inside the FPGA to limit the data transmission overhead between the FPGA and Processor as the Q-function values of all subsets are calculated inside the FPGA subsequently.

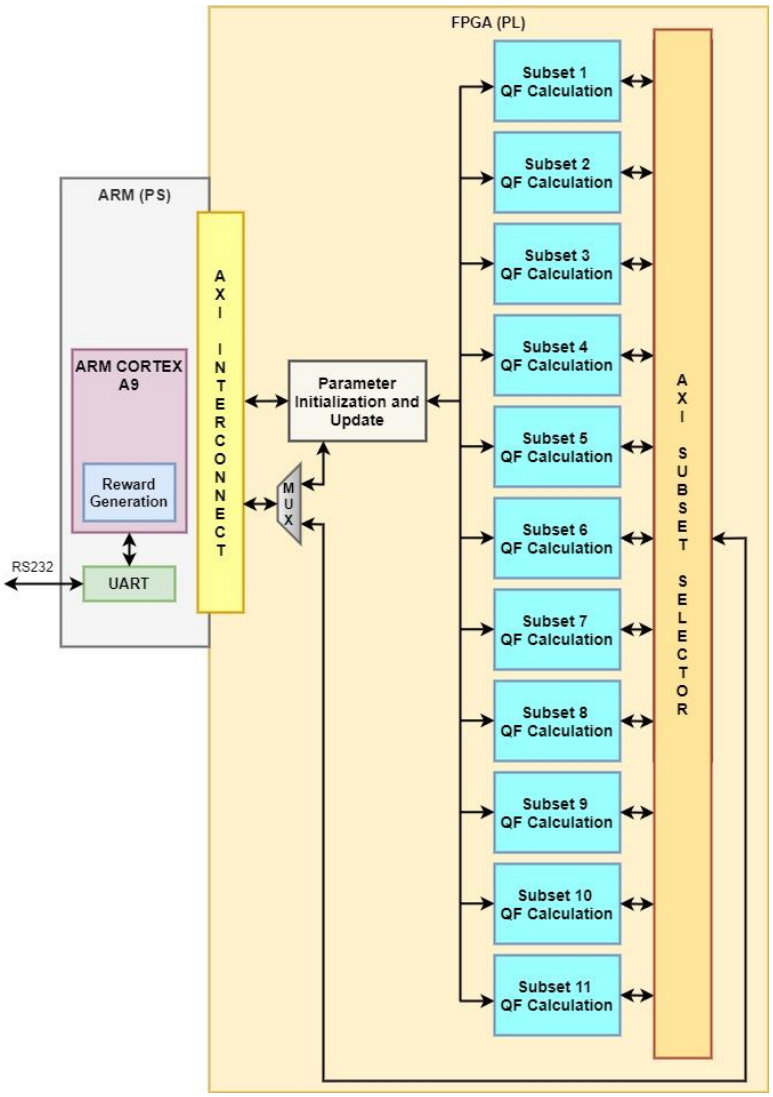
Step 4: Calculate the learned probabilities (X/T) and the Q-function values for all subsets in the MPMAB setting. This is done inside the FPGA as it is swift at complex arithmetic calculations.

Step 5: Determine the subset having the highest Q-value and then, repeat Step 1.

System Block Design



Design



Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Generating the IP for calculating the Q-function (marked as F) Step 1

1-1. Create a new project in Vivado HLS.

1-2. Select ZC706 as the board and set clock period as 20 ns.

1-3. Add CPP files for the top design and a testbench (given on GitHub). For example, top.cpp and testbench.cpp.

1-4. In the top file, write the C code to initialize and update the variables X, T and N of the machine as per the inform signal received from the lite_to_stream IP via AXI stream protocol. Then calculate the Q value (quality-factor) of the machine according to the UCB (Upper Confidence Bound) formula and pass the calculated Q-value to the comparator IP via AXI stream protocol.

1-5. Write the testbench.

1-6. Perform HLS C simulation > HLS C synthesis > C/RTL co-simulation and finally, export the RTL.

Generating the IP for comparing the Q-functions (marked as G) Step 2

2-1. Create a new project in Vivado HLS.

2-2. Select ZC706 as the board and set clock period as 20 ns.

2-3. Add CPP files for the top design and a testbench (given on GitHub). For example, top.cpp and testbench.cpp.

2-4. In the top file, write the C code to compare the 11 Q-values received from the 11 machine IPs via AXI stream protocol and pass the index of the highest Q-value machine to the processor via AXI lite protocol.

2-5. Write the testbench.

2-6. Perform HLS C simulation > HLS C synthesis > C/RTL co-simulation and finally, export the RTL.

Generating the IP for Lite-to-Stream conversion (marked as E) Step 3

3-1. Create a new project in Vivado HLS.

3-2. Select ZC706 as the board and set clock period as 20 ns.

3-3. Add CPP files for the top design and a testbench (given on GitHub). For example, top.cpp and testbench.cpp.

3-4. In the top file, write the C code to receive the 15-bit inform signal from the processor via AXI lite protocol and pass 11 different 5-bit inform signals to the 11 machine IPs via AXI stream protocol.

3-5. Write the testbench.

3-6. Perform HLS C simulation > HLS C synthesis > C/RTL co-simulation and finally, export the RTL.

Draw the system block design — Step 4

4-1. Create a new project in Vivado HLS.

4-2. Go to Settings > IP > Repository. Then, add the three generated HLS IPs to the project repository.

4-3. Click on Create Block Design. Give any suitable name to the block design.

4-4. Click on Add IP or press CTRL+I to add each IP to the block design.

4-5. Add the ZYNQ7 Processing System (marked as A). Click on the Run Block Automation button.

4-6. Add the lite_to_stream IP (marked as E). Click on Run connection automation pop-up.

4-7. After this, the AXI Interconnect (marked as C) and the Processor System Reset (marked as B) will get created automatically.

4-8. Add 4 instances of the machine IP (marked as F). Then, click on the Run Connection Automation button.

4-9. Add the Comparator IP (marked as G). Click on Run connection automation pop-up.

4-10. Add the Constant IP (marked as D). It always gives a constant 1-bit value of 1.

4-11. Connect this constant 1 value to the ap_start port under the ap_ctrl drop-down of all the HLS IPs.

4-12. Make all other appropriate connection. Then, click on Validate Design > Save Block Design.

Generating the bitstream — Step 5

5-1. Right-click on the block design name present under the Sources tab and click on Create HDL Wrapper. Then select Let Vivado manage wrapper and auto-update.

5-2. Again right-click on the block design name present under the Sources tab and click on Generate Output Products.

5-3. Click on Generate Bitstream.

Generating the baremetal application — Step 6

6-1. Go to File > Export > Export Hardware.

6-2. Check the Include Bitstream tab and click OK.

6-3. Click on File > Launch SDK.

6-4. In the SDK window, click on File > New > Application Project.

6-5. Click on Next > Hello World Application > Finish.

6-6. Write the code to run the algorithm (given on GitHub).

Enabling the NEON co-processing unit (optional) Step 7

7-1. Right-click on the name of the project under the Project Explorer tab.

7-2. Go to C/C++ Build Settings > Optimization > Set Optimization Level as Optimize most (-O3).

Running the baremetal application Step 8

8-1. Click on Program FPGA.

8-2. Open any terminal (say Tera Term). Select Serial and click OK.

8-3. Go to Setup > Serial port > Select Baud Rate as 115200 and then, click OK.

8-4. Click on Run As > Launch on Hardware (GDB).

8-5. The results of execution of the algorithm will be printed on the terminal, as shown below:

```
Subset-Learning Algorithm
**************************

Starting new experiment for 10000 time slots with
- Mean arm probabilities 0.10, 0.30, 0.50, 0.70
- Minimum subset size of 2

The subset with arms 1, 2 was selected 49 times
The subset with arms 1, 3 was selected 78 times
The subset with arms 2, 3 was selected 163 times
The subset with arms 1, 2, 3 was selected 114 times
The subset with arms 1, 4 was selected 96 times
The subset with arms 2, 4 was selected 258 times
The subset with arms 1, 2, 4 was selected 204 times
The subset with arms 3, 4 was selected 321 times
The subset with arms 1, 3, 4 was selected 669 times
The subset with arms 2, 3, 4 was selected 7526 times
The subset with arms 1, 2, 3, 4 was selected 521 times

ARM took 52183.71 us
ARM+NEON took 40249.57 us
ARM+FPGA took 10391.98 us
ARM+FPGA+NEON took 8233.12 us
```

Figure: Tera Term snapshot of the output of the baremetal application.