ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

Abstract

Artificial neural networks are one of the tools that contains computationally intensive operations. They can be utilized in many diverse areas such as detecting faults, solving financial issues, enhancing medical solutions, manufacturing applications, and even improving the quality of our daily life. Besides, neural networks are now used quite frequently in autonomous vehicles and low-power edge devices. There are many limitations such as energy consumption and performance requirements in these systems. High-speed is an important constraint in producing output within autonomous systems and energy is another main constraint, thus it should also be taken into consideration in design phase. Therefore, considering all these limitations, it is a bad choice to implement artificial neural networks in conventional Von-Neumann style processor-based embedded systems. In this project, our main objective is to build an autonomous self-driving model car that solves parking problem with using deep Q learning algorithm on FPGA. The model car will first detect the parking lot while it is roaming around the parking area and then perform the parking process. There are two different neural networks that are employed to solve this problem. First neural network is responsible to navigate the car safely in a parking area by using sensor data. An image processing algorithm that works synchronously with the first model for determining if there is a suitable parking lot for the car by processing images captured through a camera. The second neural network performs the parking process. Each neural network is simply based on deep Q learning and has different reward mechanism. They are all trained in a simulation environment which is very close the physical environment that is realized for our model car. We designed special hardware for these artificial neural networks and that will accelerate the calculations required by these neural networks in order to produce output fast enough and power efficient which cannot be achieved by a processor-based implementation. In this project, we designed a system-on-chip (SoC) architecture that contains our neural network accelerators and main controller being implemented on an ARM processor. The SoC is prototyped on Zynq-based programmable SoC board, ZedBoard. Hardware architectures that we designed is tailored to the specific needs of the neural networks and carried out the necessary computations concurrently at lower frequencies to attain higher performance than the processor-based solution. This way energy consumption is minimized and eliminate the waste of energy. The project has sections that need to be controlled and managed on the processor (ARM processor of Zynq) such as collecting the sensor data, controlling the accelerators, managing the inputs and outputs of the accelerators.

1. Introduction

Embedded system technologies have become a very popular research area with the increasing use of autonomous systems and IoT (Internet of Things) devices. Supplying power is one of the most important challenge in embedded systems. Some solutions become necessary for the energy usage. These energy problems are tried to be solved with low power modes of microprocessors or with special hardware designs as in this study. Especially, for applying the artificial neural networks successfully in autonomous systems requires efficient solutions in terms of energy consumption and processing performance since the artificial neural networks are compute intensive operations. Despite the computational density, thanks to the multi-core structure of the GPUs, high processing power results can be obtained with parallel calculations. However, FPGA-based artificial neural network special hardware designs have been shown in the literature that they can be 10 times more efficient than GPU-based solutions in terms of both energy consumption and speed performance [4].

Creating hardware design for complex and computationally intensive algorithms such as artificial neural networks using low level hardware description languages such as VHDL, Verilog is very inefficient in terms of time and effort. In this study, it is aimed to use all feasibility of High-Level Synthesis and SoC (on-chip system) architecture. It is desired to create an optimized system both by using the correct design, without the need for low-level languages such as VHDL, Verilog during the hardware design phase, and by performing the stages that are more advantageous with the processor with the ARM processor.

The main purpose of the project is to ensure that the prototype can park autonomously in a suitable place in the car park by obtaining energy saving and high process performance with special hardware designs. Thanks to 3 distance sensors and 1 camera to be used, the possibility of the car hitting any objects or assets is eliminated. In the designed project, the car can find a parking space in a fully autonomous manner without any human element and can perform the parking process.

Many studies have been done on this subject and there are ongoing studies [1]. Some of these studies focus directly on the application of reinforcement learning technique [2]. In such studies, factors such as energy consumption and computing power are ignored. The Reinforcement learning algorithm is run on a development board such as jetson nano, raspberry pi, without any performance or energy concerns [3]. Some studies have focused on creating efficient hardware designs for artificial neural networks. Low level hardware definition languages such as VHDL and Verilog were generally used in these studies. In some of these studies, the backpropagation stages of neural networks have also been accelerated [4]. Application specific Hardware designs provides huge advantages in many areas, not only in autonomous vehicles, and especially in image processing algorithms which are based on convolutional neural networks[5] [6]. In this study, reinforcement learning was worked on both the software side and creating special hardware design.

With this project, it is aimed to increase the reliability of driving and the prevalence of autonomous vehicles by reducing the accident rates of vehicles during parking, and to realize this process in a practical, energy-efficient, and high process performance. It is also aimed to solve the problems brought by the parking event, which is a part of the traffic, and to show the benefits of the special hardware design for autonomous systems. In the successful implementation of the project, it is thought that positive results such as decreasing accident rates, decreasing costs due to accident, saving time, decreasing fuel consumption, psychological relief and obtaining low energy high operation performance in embedded systems will be seen.

The main contributions of this study as follows,

- Creating efficient hardware designs in terms of energy consumption and computing performance by using HLS directives of artificial neural network algorithm,
- Presenting a solution of parking problem in simulation environment with deep q learning,
- Creating necessary software/hardware designs and testing artificial neural network accelerators on prototype car.

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

1.1. Marketability/Innovation

Autonomous systems and artificial intelligence applications are a part of our lives. Today, we can see autonomous systems and artificial intelligence designs in many fields when used together. Especially in the automobile industry, companies spend a lot of time and money for autonomous driving. Tesla , one of the leading companies in the electric vehicle industry, also designs equipment for the applications which is developed [7]. It is seen that a company that pioneers the sector gives importance to application-specific hardware design. The hardware design techniques used in this study can be used not only in artificial intelligence applications, but also in the hardware design of any algorithm. With the spread of IoT technology, the number of devices powered by portable power supply has increased. Thus, the need of algorithms for energy efficient special hardware designs has increased. Therefore, this study contributes to such needs in terms of solutions and ideas.

1.2. Reusability

This project contains research topics that are very popular today. Artificial neural networks are used quite frequently both in reinforcement learning algorithms and in computer vision. It is very useful to use in many areas due to the high accuracy rates neural networks provide. In addition to these benefits, they need high processing performance. As a solution to this, GPUs have started to be used for accelerating the neural networks. Even if the process performance approaches the desired level, energy consumption remained as a big problem. Similar studies show that the most effective solution for energy consumption is application-specific hardware designs [8][9]. In this study, fully connected models with different architectures is used. The HLS pragmas that are used in the hardware design of the model will be easily applicable to another model. Since the project includes FC models, it is presented how to improve the hardware design of these architectures. Based on this, models with different number of layers and neurons will be able to easily achieve efficient designs using the same techniques. In addition, source codes related to camera, motor control and distance sensor, which are frequently needed in this kind of studies, have been created. In the design created for the camera, the images were not stored, but the ports, which allow us to store pixel information, were not removed. Besides, block design is reusable in similar projects. The designs for created motor control and reading sensor data can reuse with using the IP repository. All designs used in the project have been designed for reuse. Neural network designs are very useful even for artificial neural networks with different architecture.

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

2. Method

Figure 1 shows the flow chart of the project. Detailed information about the main stages will be given in the following sections.

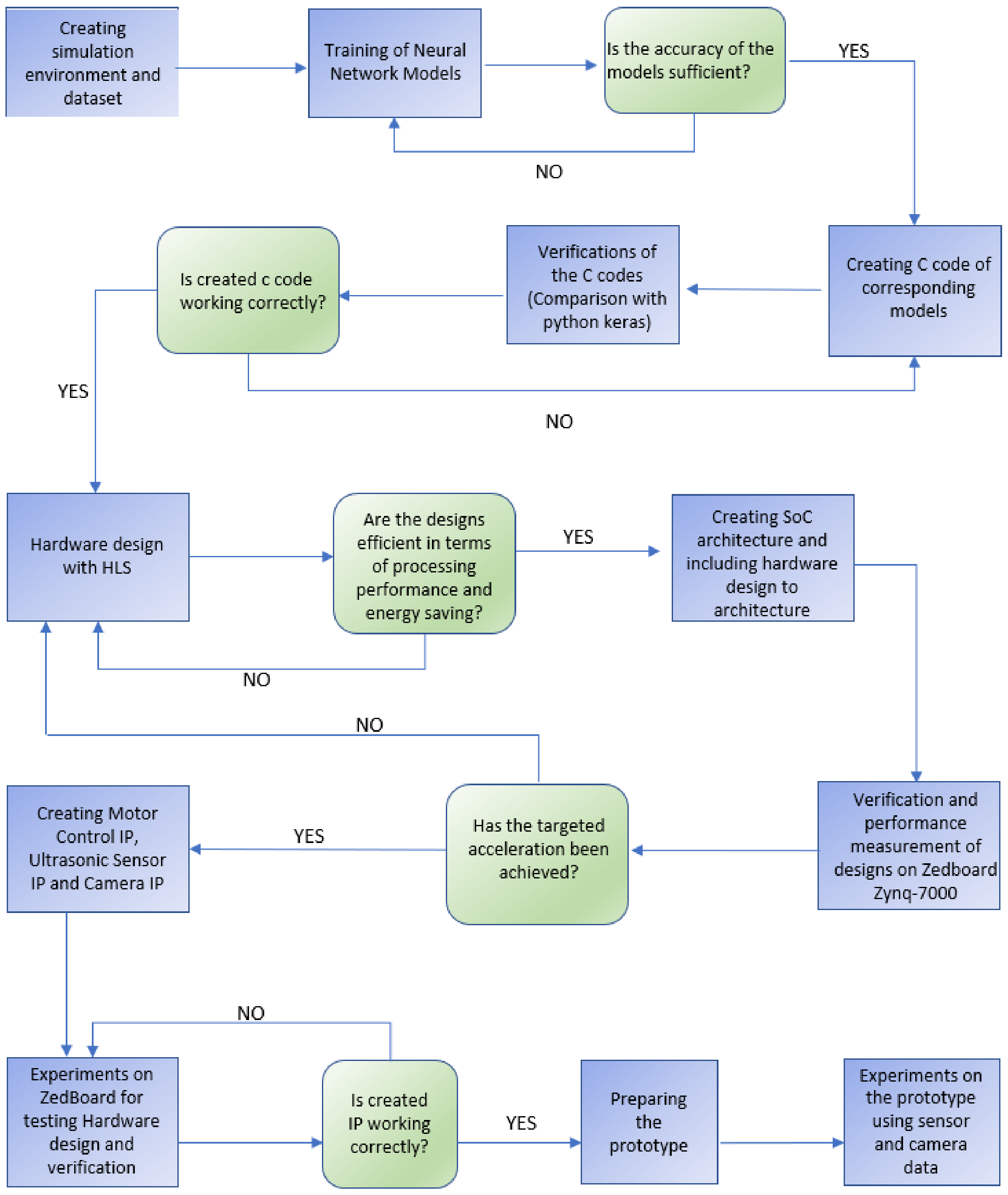
Creating simulation environment and dataset
Training of Neural Network Models
Is the accuracy of the models sufficient?
YES
NO
Creating C code of corresponding models
Verifications of the C codes (Comparison with python keras)
Is created c code working correctly?
YES
NO
Hardware design with HLS
Are the designs efficient in terms of processing performance and energy saving?
YES
NO
Creating SoC architecture and including hardware design to architecture
Verification and performance measurement of designs on Zedboard Zynq-7000
Has the targeted acceleration been achieved?
NO
YES
Creating Motor Control IP, Ultrasonic Sensor IP and Camera IP
Experiments on ZedBoard for testing Hardware design and verification
Is created IP working correctly?
NO
YES
Preparing the prototype
Experiments on the prototype using sensor and camera data

Figure 1: Flow Chart

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

2.1. Simulation

Simulation is required to apply the deep q learning method. The agent should solve the problem by experimenting. The learning phase can take a very long time. It is almost impossible to train the neural network from scratch in real environment. In order for the artificial neural network trained in the simulation environment to achieve the same output in the real environment, the simulation must be compatible with real-world physics and scalable. Considering this situation, the simulation environment was designed entirely by the team and the vehicle's motion capabilities and scaling were easily adjusted for real prototype. Figure 2 and 3 shows the real environment and the simulation environment created by the team. The simulation environment was prepared using the Python based Pygame module.

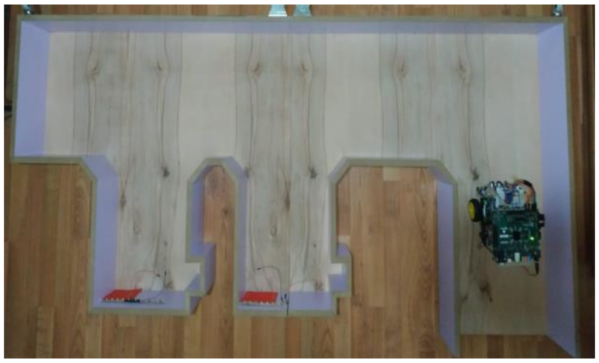
Figure 2: Real environment

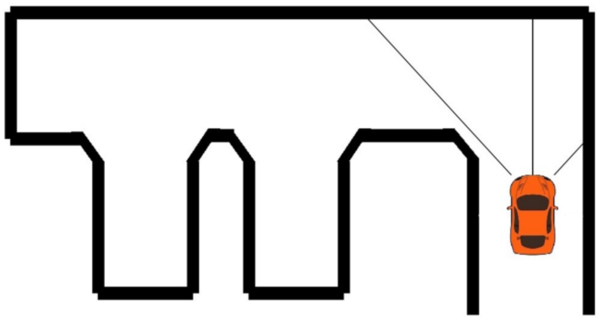
Figure 3: Simulation environment

2.2. Deep Q Learning

Deep q learning has the following 2 main objects; An agent trying to solve the problem, an environment where selected actions can be applied. The relationship between these two main objects is shown in Figure 4. The process of Q-Learning creates a knowledge for the working agent which it can refer to maximize its reward in the long run. Knowledge accumulation is stored in a table in Q learning, but in this method the vehicle must have tried all possible situations beforehand. In deep q learning, an artificial neural network is used to build up knowledge and artificial neural network can make correct decisions for any state even if it has not been tried before. Therefore, it is very important to use neural networks in such applications. Python based Keras module was used for artificial neural networks to be used in deep q learning algorithm. Keras module is very useful thanks to its high-level interface with neural networks. During the training, the score system was developed to observe the learning of the model. The agent increased his score in every action which is not wrong. Neural network models of agents that have passed score limit was saved. The behavior (actions for any state) of the saved models were checked manually. The models used in the project are given in Table 1. Since the parking task is more complicated than moving without collision in the parking area, the neural network which is used for parking has more neurons.

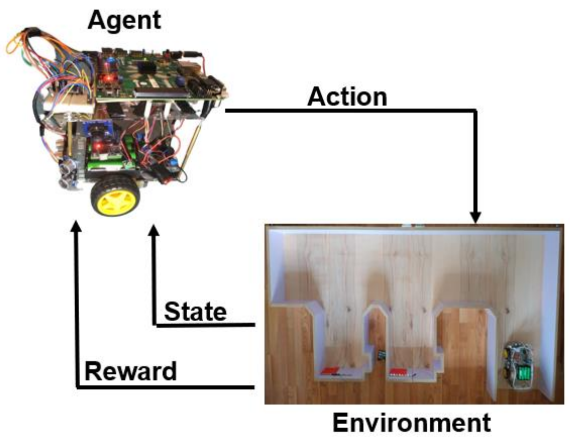

Figure 4: Deep Q Learning Algorithm

2.3. Training the Neural Network Models

Different artificial neural network models were used to complete different tasks in the project. One of these models will allow the vehicle to move in the parking lot until the suitable parking area is found. The other will ensure that the parking process is completed in the determined parking area. For the training of these two models, reinforcement learning technique was used. As can be seen in the Figure 1, a simulation environment has been created to carry out these trainings. Header files were created by taking the weights of these trained models. C codes containing forward feeding algorithms of these models were created and trained weights were used in this algorithm. For the verification of the generated C code, layer by layer comparisons were made with the model in the Python-Keras environment. Hardware design of the verified C codes were created with Vivado HLS.

2.4. Designing Hardware Accelerator with HLS

The most critical stage of this work is the creation of efficient hardware design in terms of process performance and energy saving. Low level hardware definition languages such as VHDL, Verilog are very costly in terms of effort and time to perform hardware design. In this study, it will be shown that designs with efficiency close to hardware created with low level languages can be made using HLS. In order to create efficient hardware designs, the algorithm must be written HLS friendly. In addition to this, it is necessary to provide information with HLS pragmas about how hardware design should be. In this project, HLS PIPELINE pragma which is one of the most common and powerful pragma was frequently used. Figure 5 is taken from Xilinx website and it shows the effect of pipeline pragma for hardware design. The PIPELINE pragma reduces the initiation interval for a function or loop by allowing the concurrent execution of operations. The second pragma that we used in this project is ARRAY RESHAPE pragma. This pragma creates a new array with fewer elements but with greater bit-width, allowing more data to be accessed in a single clock cycle. There is a trade-off between the performance increase of the design and the increase in resource utilization. Since the performance increase can be achieved with the using more resources, the resources of the development card (In this study ZedBoard Zynq-700) creates constraints. Since multiple accelerators were used in this study, it was necessary to balance between resource usage of accelerators.

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

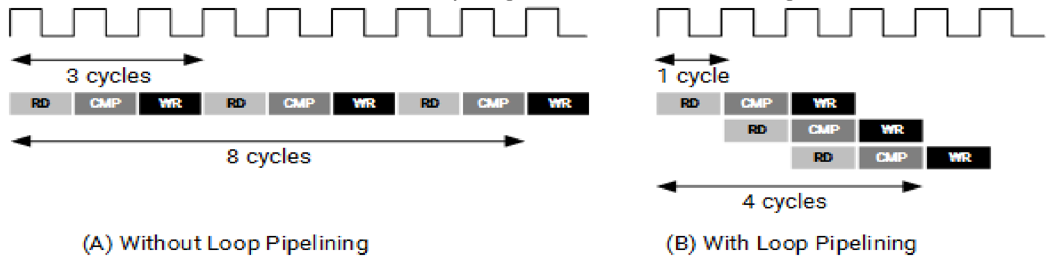

Figure 5: Effect of PIPELINE Pragma

2.4.1 FC Neural Network for Searching Available Parking Lot

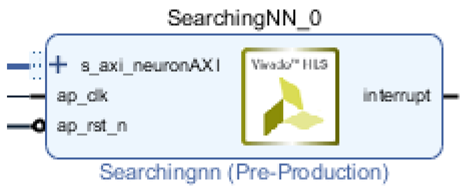

Figure 6: SearchingNN block design

Considering the model in FC Model – 1 experiment whose results are presented in the table 1, it was decided to create a smaller model as seen in Figure 6 for the searching neural network which will allow the vehicle to move without crashing in the parking area. The model with the neuron structure of 3-24-48-24-3 was successful in the training part. Header files were created for the weights and biases of this model to be used in hardware design. The following algorithm shows the fully connected processing unit between layer 3 (24 neurons) and layer 4 (48 neurons). The algorithm was accelerated using the PIPELINE pragma. It was used with different initiation intervals in the same way in other processing units. In this section, initiation interval is selected as 4. This value was determined by conducting experiments on the balance between performance increase and resource usage. Sources to be used by other designs were taken into consideration while designing.

```
for(i=0;i<24;i++){
#pragma HLS PIPELINE II=4
    sum=0;
    for(j=0;j<48;j++)
        sum+=Layer3_neurons[j]*Layer4_weights[i][j];
    Layer4_neurons[i]=(sum+Layer4_bias[i])>0?(sum + Layer4_bias[i]):0;
}
```

2.4.2 FC Neural Network for Parking Process

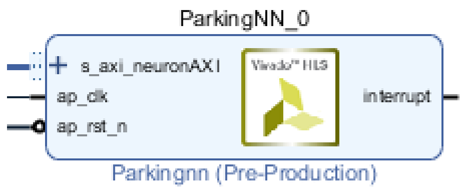

Figure 7: ParkingNN block design

For the model that will perform the parking process, we have chosen a model in Figure 7 that contains more neurons than the search model, since parking is a more complicated problem than searching. The model with the neuron structure of 3-32-64-32-4 was successful in the training part. Header files were created for the weights and biases of this model to be used in hardware design. The following algorithm shows the calculation step of the output layer. Since Bram's has 2 ports, initiation interval can be at least 16 in this algorithm when pipeline pragma is used alone. Therefore, the initiation interval was set to be 4 thanks to the ARRAY RESHAPE pragma. Sources to be used by other designs were taken into consideration while designing.

```
#pragma HLS ARRAY_RESHAPE variable=Layer4_neurons block factor=4 dim=1
for(i=0;i<4;i++){
#pragma HLS PIPELINE II=4
    sum=0;
    for(j=0;j<32;j++)
        sum+=Layer4_neurons[j]*Layer5_weights[i][j];
    out[i]=sum+Layer5_bias[i];
}
```

2.5. Creating Hardware Designs for Camera

In the project, data will be taken from the camera (ov7670) for parking area detection. This camera will be connected to PMOD pins on Programmable logic side in SoC design. Hardware design is required to read data from the camera. 3 basic block designs were created to meet this need. These blocks; capture in Figure 9, controller in Figure 8, and capture to processor in Figure 9.

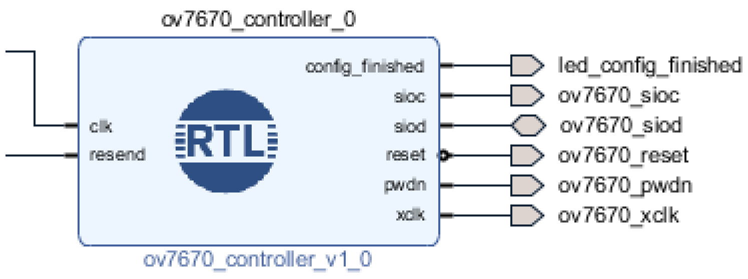

Figure 8: ov7670_controller block design

Omnivision OV7670 uses Omnivision Serial Camera Control Bus (SCCB) protocol to set up the camera parameters. SCCB is I²C-compliant interface but avoids the usage of I²C brand due to licensing fees 2. The controller component is composed of three components: I²C bus master, OV7670 instructions and glue code.

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

The capture block allows the pixel values to be read from the camera. Normally, this block also allows the captured pixels to be addressed and written to a block ram. However, since this project will be used on red object detection, no storage will be done. Block provides some information about red pixel which captured from camera.

A block with AXI interface was created to read the red pixel information provided by capture block from the processor section. Thanks to this block, communication between the capture block and the processor has been achieved.

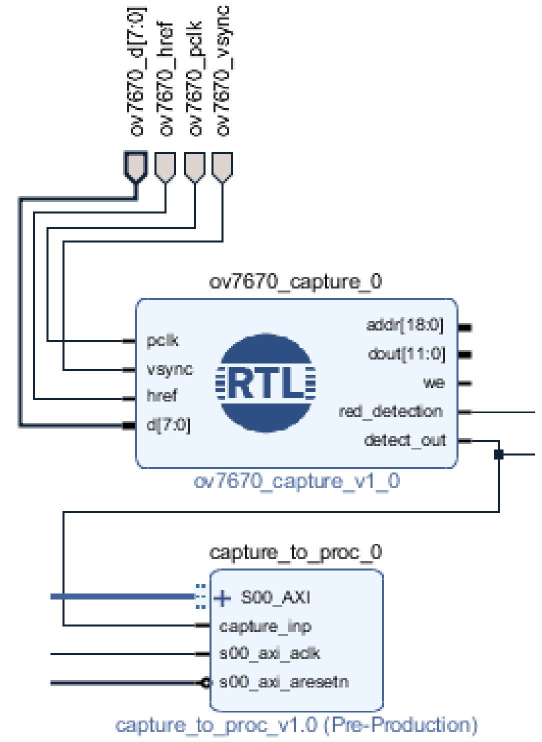

Figure 9: ov7670_capture and capture to proc block design

2.6. Creating Hardware Design for HCSR-04 Sensor

A custom IP (Intellectual property) as seen in Figure 10 was created for the distance sensor to be used in the prototype. This IP has an input "echo" and an output "trigger". Since the prototype has 3 distance sensors, 3 hcsr_04_ip was added to the block design. Trigger input must be used for at least 10us as high-level signal to make an accurate measurement. Since the PL part of ZedBoard runs at 100MHz, the counter was counted up to 1000 in order to reach 10us. Thus, HIGH signal was given to the trigger during 10us.

```
if trig_finished = '0' then
    if counter = 1000 then
        trigger <='0';
        trig_finished <='1';
        counter <=(others=>'0');
    else
        trigger <='1';
        counter <=counter+1;
    end if;
```

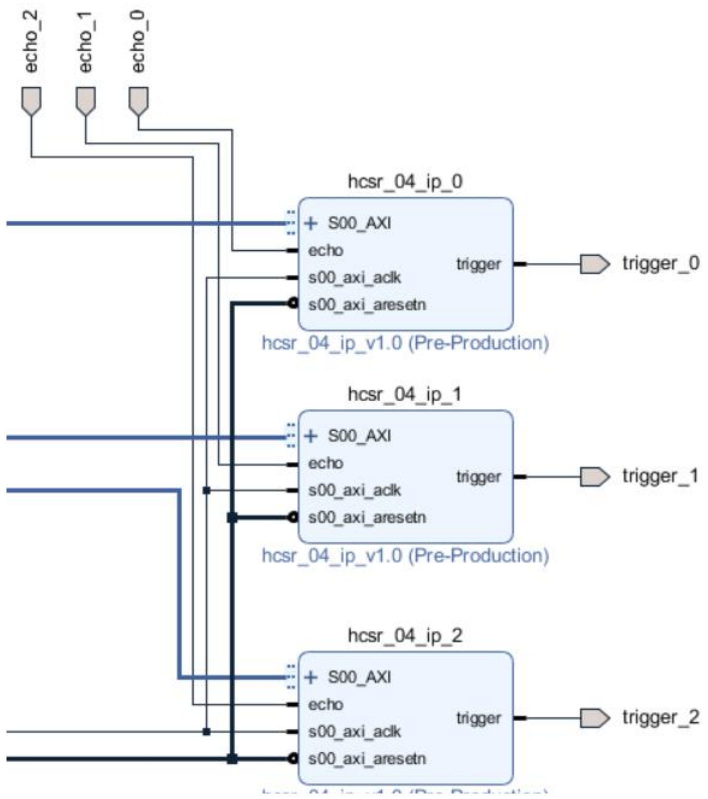

Figure 10: hcsr_04_ip block design

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

When echo signal comes back, it generates a pulse signal and pulse width is proportional to the distance between the sensor and the object next to it.

Therefore, Distance = $(pulse\ width\ of\ echo\ *\ velocity\ of\ sound\ (340\ m/s))$ / 2

In our project;

Pulse width of echo: $1/100000000\ =\ 10\ us$

Velocity of sound: $34000\ cm/s$

So, counter must be count up to: $1/[(1/100000000) * 340000)/2]\ = 5882$

It is observed how many cycles the Echo signal is HIGH. It is calculated how many 5882 there are in this cycle. Since the HCSR-04 distance sensor can measure the range of 2-4000 cm, the trig finished signal is set to LOW when the distance reaches 3500 cm. Thus, distance measurement is completed.

```
elsif counter = 5882 then
    if distance_ones = 9 then
        distance_ones <= (others => '0');
        distance_tens <= distance_tens + 1;
    else
        distance_ones <= distance_ones + 1;
    end if;
    distance <= distance + 1;
    counter <= (others => '0');
    if distance = 3500 then
        trig_finished <= '0';
    end if;
```

The last measured distance value is written to the register.

```
      case loc_addr is
        when b"00" =>
          reg_data_out<= slv_reg0;
        when b"01" =>
          reg_data_out<= slv_reg1;
        when b"10" =>
          reg_data_out<="000000000000000000000000"&std_logic_vector(saved_tens)
                        &std_logic_vector(saved_ones);
        when b"11" =>
          reg_data_out<= slv_reg3;
        when others =>
          reg_data_out<= (others => '0');
      end case;
```

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

2.7. Creating Hardware Design for Motor Control

A custom IP in Figure 11 was created for the motor control. This IP has outputs "right" and "left". Arduino was used for transmitting the corresponding action output of ZedBoard to DC motors.

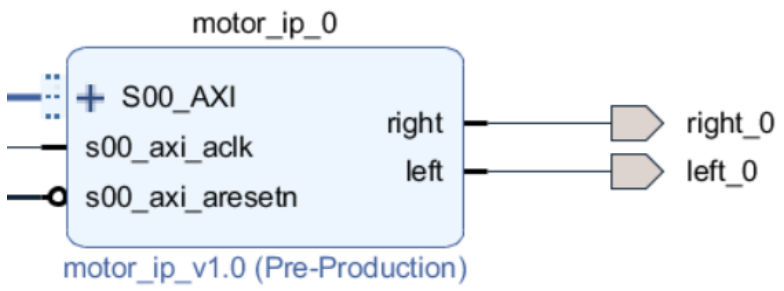

Figure 11: motor_ip block design

The last 2 bits of slv_reg1 were used for motor control.

```
process (S_AXI_ACLK)
  begin
   if rising_edge(S_AXI_ACLK) then
        right <= slv_reg1(1);
        left <= slv_reg1(0);
   end if;
end process;
```

2.8. Creating Block Design for all Components to Work Together

The connections of all components in block design have been completed. By operating all components together, the vehicle will be ensured to perform parking in the real environment. Figure 12 shows block design, which includes all components. Except for the designs created for the camera, a clock with a frequency of 100 MHz is connected to all designs. The critical path in clock period selection is neural network designs. Neural networks are designed to work well with the 10ns clock period. Thanks to AXI Interconnect block, all blocks with AXI interface on the programmable logic side can be communicated with the processor via a single M_AXI interface. Since the block, which enables reading data from the camera, does not have an AXI interface, an IP with an AXI interface named "capture_to_proc" was created and it was provided to use the information received from the camera on the processor side.

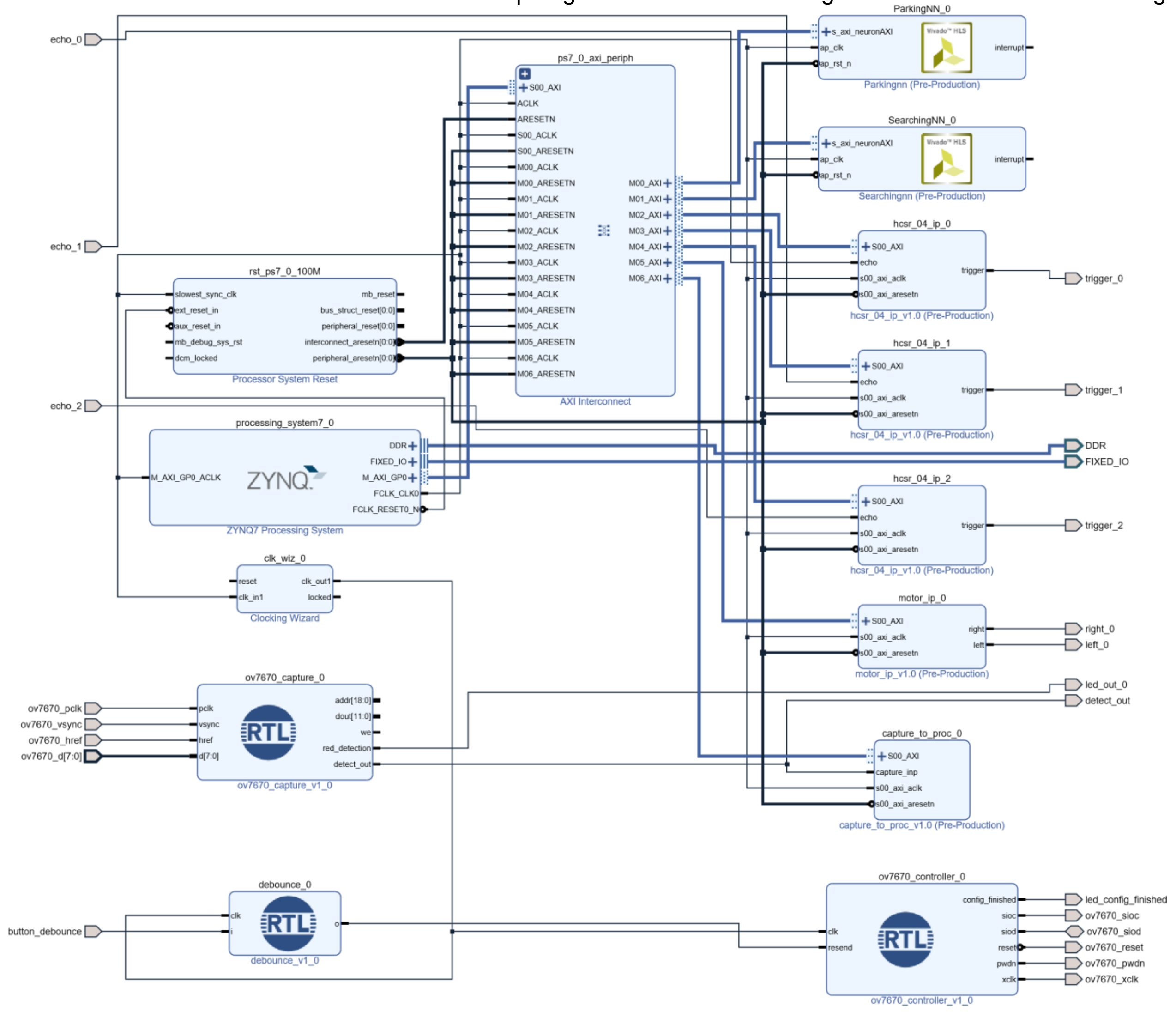

Figure 12: Block Design of Project

2.9. Software Development Kit

Created designs were checked in SDK by programming the arm processor and using the M_AXI interface. Thus, the synchronization between them was organized with SDK while the designs were working synchronously. At the same time, some processor-friendly necessary operations were made with the arm processor. In summary, the SDK was used to do the following operations:

- Control the components of design
- Handle processor-friendly processes
- Synchronizing the components

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

For reading sensor values, following function was used.

```
void readSensors(){
      readValue = HCSR_04_IP_mReadReg(0x43C20000,
      HCSR_04_IP_S00_AXI_SLV_REG2_OFFSET);
      while(readValue == 0){
            readValue = HCSR_04_IP_mReadReg(0x43C20000,
            HCSR_04_IP_S00_AXI_SLV_REG2_OFFSET);
      }
      ones   = readValue & 0x0000000f;
      tens    = (readValue & 0x000000f0) >>4;
      distance_right = tens*10 + ones;

}
```

For the model that will perform the searching process, following function was used in SDK.

```
void searchingNN(){
      searchingNN_in[0] = state[0];
      searchingNN_in[1] = state[1];
      searchingNN_in[2] = state[2];

      searchingNN_control[0] |=0x1;
      while((searchingNN_control[0]&0x2) != 0x2);

      printf("%f %f %f \n",
      searchingNN_out[0],searchingNN_out[1],searchingNN_out[2]);
      if(searchingNN_out[0] > searchingNN_out[1] && searchingNN_out[0] >
      searchingNN_out[2]){
            printf("forward\n");
            forward();
            parking_lot();
            stop();
      }
      else if(searchingNN_out[1] > searchingNN_out[0] && searchingNN_out[1] >
      searchingNN_out[2]){
            printf("left\n");
            turnLeft();
            parking_lot();
            stop();
      }
      else{
            printf("right\n");
            turnRight();
            parking_lot();
            stop();
      }
}
```

For the model that will perform the parking process, following function was used in SDK.

```
void parkingNN(){
      parkingNN_in[0] = state[0];
      parkingNN_in[1] = state[1];
      parkingNN_in[2] = state[2];
```

```
        parkingNN_control[0] |=0x1;
        while((parkingNN_control[0]&0x2) != 0x2);

        printf("%f %f %f %f\n",
        parkingNN_out[0],parkingNN_out[1],parkingNN_out[2],parkingNN_out[3]);
        if(parkingNN_out[0] > parkingNN_out[1] && parkingNN_out[0] >
        parkingNN_out[2] && parkingNN_out[0] > parkingNN_out[3]){
                printf("forward\n");
                forward();
                parking_lot();
                stop();
        }
        else if(parkingNN_out[1] > parkingNN_out[0] && parkingNN_out[1] >
        parkingNN_out[2] && parkingNN_out[1] > parkingNN_out[3]){
                printf("left\n");
                turnLeft();
                parking_lot();
                stop();
        }
        else if(parkingNN_out[2] > parkingNN_out[0] && parkingNN_out[2] >
        parkingNN_out[1] && parkingNN_out[2] > parkingNN_out[3]){
                printf("right\n");
                turnRight();
                parking_lot();
                stop();
        }
        else{
                printf("stop\n");
                parking_is_done=1;
                stop();
        }
}
```

2.10. Technical Complexity

The study consists of 2 main stages. The first step is to train the artificial neural network models to be used in the software environment with the Python-Keras module. Even if the main purpose of the study is to produce high performance artificial neural network accelerators, it was aimed to demonstrate this system by solving a real problem with the prototype car. Therefore, a simulation environment which have the same physical features with real world was created for the training section of neural network models. The reinforcement learning technique was used in training session, and the most important parameter of this method is the reward mechanism. Many experiments were done to reach the correct parameters. The models that gained the ability to solve the problem were saved and their weights were taken to the header files and moved to the second stage. This stage includes the main purpose of the project. It is very important to create and verify the C code of the model. Besides being-high performance the hardware is expected to provide the correct output. The hardware design phase could also be accomplished with low level hardware definition languages such as VHDL, Verilog. However, HLS synthesis was preferred instead of VHDL or Verilog since it would be quite efficient in terms of time and effort. In hardware design with HLS, how to transfer C code to hardware design should be explained using HLS pragmas to compiler. At this stage, the effect of pragmas on design should be evaluated by experiments. Another issue to be considered at this stage is the use of resources. Multiple accelerator designs are made in a system with limited resources and increased performance often results in increased resource utilization. The balance between performance and resource utilization should be well established. If the increase in resource usage cannot provide sufficient performance increase, different design techniques should be tried, and the most optimized system should be created.

3. Empirical Setup

In this study, simulation was created by using Python-Pygame module to implement reinforcement learning technique. The scaling of the simulation environment was done taking into account the dimensions of the prototype. Likewise, the movement capabilities of the vehicle in the simulation environment were designed in accordance with the prototype. C codes created with trained weights will be tested with GNU compiler in linux operating system. Hardware designs of verified C codes and IPs of the designs will be created using Vivado HLS 2017.4. SoC architecture will be created using Vivado Design Suite 2017.4. Zynq processor and the hardware accelerators created with HLS will be included in the architecture. The created SoC architecture will be implemented on the ZedBoard Zynq-7000 card and will be tested on the prototype after the outputs are verified. Hcsr-04 Sensor IP and Motor Control IP were created, and they were tested on the prototype. The TXS0108E 8 Channel Logic Level Converter was used when connecting the distance sensors and Arduino to ZedBoard. Because ZedBoard' s pins give 3.3V, however Arduino and distance sensors use 5V. Arduino is set to move the motors according to the data from the 2 pins of the Zedboard. The main code in the content of Arduino is as follows;

```
void loop() {
  Zedboardright= digitalRead(Zedboardr);
  Zedboardleft= digitalRead(Zedboardl);
  if ( Zedboardright==HIGH and Zedboardleft==HIGH){
      forward();
  }
  else if ( Zedboardright==HIGH and Zedboardleft==LOW){
      right();
  }
  else if ( Zedboardright==LOW and Zedboardleft==HIGH){
      left();
  }
  else{
      dmove();
  }
}
```

A prototype model was prepared in order to make experiments which are related about sensors and motors. The experiments which is related camera were also completed using ZedBoard and camera. In the next step, the camera will be added to the prototype. Here you can see the pictures of the prototype model in Figure 13.

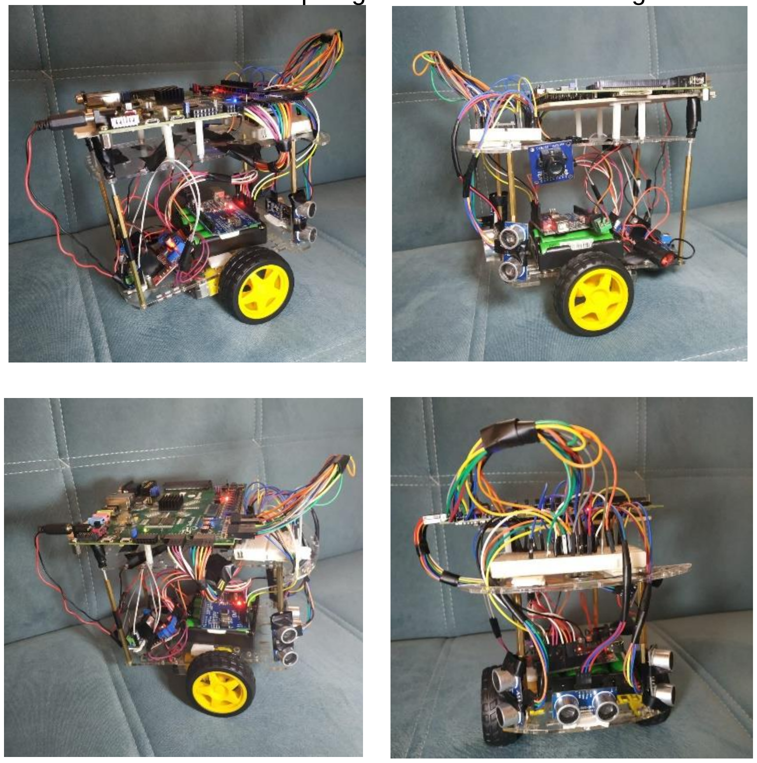

Figure 13: Photos of prototype

4. Empirical Results

The models trained with the Reinforcement learning technique were visually tested. Failed models were tried again by changing the reward mechanism. Data set consisting of approximately 1200 image was created to be used in the training of the CNN model to be used in parking lot determination. The CNN was trained with this data set and accuracy was recorded as 86% as it appears in the Figure 14. In real experiments, the CNN model could not reach the accuracy as good as it reached on the verification data. Besides, the hardware designs created for the CNN model are quite inefficient in terms of resource use. Therefore, instead of using the CNN model, it was decided to make an image processing application that can detect the red object.

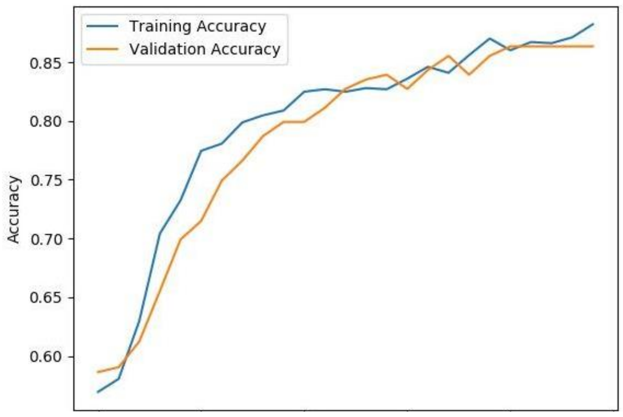

Figure 14: Accuracy of CNN Model

C code was created for model with 3 hidden layers, which were trained in the simulation environment. Hardware design was realized with Vivado HLS. Table 1 presents the execution time of this model on the ARM processor and hardware design. The hardware design, which operates at a much lower frequency(100MHz) than the ARM processor(667MHz), has managed to produce output approximately 17 times faster. It is very important to reach this speed in low frequency in order to create energy efficient design. In order to create another perspective, the execution time of the model was measured on the Python-Keras module where we perform model verification. The model produced output at 10.4 µs on Python-Keras (Intel i7-6700HQ with 2.6GHz 4-cores 8-threads processor). This shows that, higher processing performance is achieved than a Keras module which is highly optimized module working on a hard silicon processor. Based on the acceleration and resource usage of the first experiment (FC Model - 1), the size of the original models to be used in the project was decided. The pragmas used in the first experiment were used in FC Model 2 and 3. The model (FC Model - 2), which will provide control of the vehicle until the parking area is found, has reached sufficient acceleration as seen in table 1. Besides hardware design of Searching Model is very suitable in terms of resource use as seen in table 2. The model (FC Model - 3) that will perform the parking process contains more neurons than the searching model. It was expected to use more resources than the searching model to achieve the same acceleration rate. Nevertheless, design have not required extra optimization since it used an acceptable amount of resources. The outputs produced by the models were checked and verified that the synthesis process was completed successfully. Since the captured pixel data is not stored in the memory, the resource usage of the designs created for the camera does not have a limiting effect for other designs. Red object detection was tested by reading data from the camera and verified.

Table 1: Execution Times of Models on Two Platform

Neural Network Models	**Execution Time on Zynq ARM Processor**	**Execution Time on Hardware Design**
Fully-Connected Model – 1 3-64-128-64-3	725.69 µs	43.52 µs
Fully-Connected Model – 2 Searching 3-24-48-24-3	121.27 µs	8.80 µs
Fully-Connected Model – 3 Parking 3-32-64-32-4	193.74 µs	11.52 µs

As seen in Table 2, the resource usage of the hardware designs which were created for the motor control, HCSR-04 sensor is very low amount. Because the values are taken with a single register. A single register was used for reading the ones digit and the tens digit of the measured distance value in HCSR-04 Sensor IP. Similarly, a single register was used in order to control left and right movements in Motor Control IP.

If the sensors were software based, the distance values would be read in order, so this would cause delay. In our project, all three sensors can measure simultaneously. In addition to this, while the sensors are measuring the distances, the camera can also collect data at the same time.

Table 2: Resource Utilization of Hardware Designs

Hardware Designs	**LUT**		**DSP48**		**BRAM36**	
	Used	**Util%**	**Used**	**Util%**	**Used**	**Util%**
Fully-Connected Model – 1 3-64-128-64-3	20701	38.91	35	15.91	70.5	50.36
Fully-Connected Model – 2 Searching 3-24-48-24-3	14588	27.42	62	28.18	15.5	11.07
Fully-Connected Model – 3 Parking 3-32-64-32-4	18760	35.26	82	37.27	56	40
Hardware Designs for Camera (capture, controller, debounce, capture to processor)	471	0.89	0	0	0.5	0.36
Hardware Designs for HCSR-04 Sensor	103	0.19	0	0	0	0
Hardware Designs for Motor Control	57	0.11	0	0	0	0
The Block Design with All Component	33759	63.46	144	65.45	75	53.57

In this study, resources of ZedBoard have been used to a great extent. Figure 15 shows the usage of resources on the device scheme. The capacity of the ZedBoard is used 90% in terms of slice. Worst Negative Slack (WNS) value is 0.438 ns. This shows that ZedBoard is used efficiently in the project. The amount of wasted resources is very low.

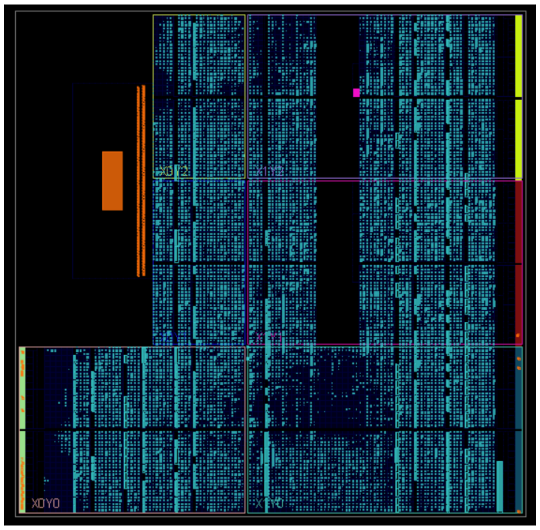

Figure 15: Usage of Resources on Device Scheme

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

5. Conclusion

In this project, hardware accelerators are designed with high-level synthesizer for different artificial neural networks. It is shown that faster and energy efficient results are obtained as a result of easier design processes compared to low level hardware description language thanks to specially created hardware. Operating at a much lower frequency (100MHz) than the ARM processor (667MHz), the Vivado HLS-based hardware design produces up to 17 times faster output. The resource use of the hardware design for the model that will allow the vehicle to navigate and enter the parking lot is given in Table 2. It is seen that the use of BRAM is higher than other components. It has been observed that the use of BRAM will increase in optimized hardware designs. Based on this, Parking problem were tried to be solved without using very large models in terms of number of neurons it contains. For large neural network models, design difficulties increase if their hardware is designed using languages such as VHDL and Verilog. This study has showed the benefits of application-specific hardware design in terms of speed and energy savings. It has been observed that the high-level synthesis makes the design phase much easier. Autonomous parking has been successfully completed on the SoC architecture.

ELON: Autonomous Parking via Deep Q Learning: A SoC Solution

List of group members: Alican Özeloğlu
İsmihan Gül Gürbüz
Team Stark
Github Link: https://github.com/AlicanOzeloglu/ELON-Autonomous-Parking

References

[1] N.-S. Huang, J.-M. Braun, J. C. Larsen, ve P. Manoonpong, "A scalable Echo State Networks hardware generator for embedded systems using high-level synthesis", içinde *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, Haz. 2019, ss. 1-6, doi: 10.1109/MECO.2019.8760065.

[2] Q. Zhang, T. Du, ve C. Tian, "Self-driving scale car trained by Deep reinforcement Learning", s. 6.

[3] K. Tiba, R. M. Parizi, Q. Zhang, A. Dehghantanha, H. Karimipour, ve K.-K. R. Choo, "Secure Blockchain-Based Traffic Load Balancing Using Edge Computing and Reinforcement Learning", içinde *Blockchain Cybersecurity, Trust and Privacy*, c. 79, K.-K. R. Choo, A. Dehghantanha, ve R. M. Parizi, Ed. Cham: Springer International Publishing, 2020, ss. 99-128.

[4] F. Ortega-Zamorano, J. M. Jerez, D. Urda Munoz, R. M. Luque-Baena, ve L. Franco, "Efficient Implementation of the Backpropagation Algorithm in FPGAs and Microcontrollers", *IEEE Trans. Neural Netw. Learn. Syst.*, c. 27, sy 9, ss. 1840-1850, Eyl. 2016, doi: 10.1109/TNNLS.2015.2460991.

[5] Yongmei Zhou ve Jingfei Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks", içinde *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, Harbin, China, Ara. 2015, ss. 829-832, doi: 10.1109/ICCSNT.2015.7490869.

[6] W. Farhat, H. Faiedh, C. Souani, ve K. Besbes, "Real-time embedded system for traffic sign recognition based on ZedBoard", *J. Real-Time Image Process.*, c. 16, sy 5, ss. 1813-1823, Eki. 2019, doi: 10.1007/s11554-017-0689-0.

[7] E. Talpes *vd.*, "Compute Solution for Tesla's Full Self-Driving Computer", *IEEE Micro*, c. 40, sy 2, ss. 25-35, Mar. 2020, doi: 10.1109/MM.2020.2975764.

[8] E. Del Sozzo, A. Solazzo, A. Miele, ve M. D. Santambrogio, "On the Automation of High Level Synthesis of Convolutional Neural Networks", içinde *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, USA, May. 2016, ss. 217-224, doi: 10.1109/IPDPSW.2016.153.

[9] K. Guo, S. Zeng, J. Yu, Y. Wang, ve H. Yang, "A Survey of FPGA-Based Neural Network Accelerator", *ArXiv171208934 Cs*, Ara. 2018, Erişim: Nis. 17, 2020. [Çevrimiçi]. Erişim adresi: http://arxiv.org/abs/1712.08934.