Hardware/Software Codesign for a Fuzzy Autonomous Road-Following System

Yi Fu, Howard Li, *Senior Member, IEEE*, and Mary E. Kaye

***Abstract*—In this research, a fuzzy logic controller is designed for vision-based autonomous road-following. Because of its high-speed response, portability, and flexibility, a field programmable gate array is applied to implement this control system. Furthermore, a novel hardware/software partitioning method using the genetic algorithm is developed. This method is capable of finding the tradeoff among several evaluation factors under conditions of hard constraints. A small-scaled intelligent vehicle, which is capable of autonomous road-following is built and the proposed controller is tested in the real-world environment. Experiments of hardware implementation and codesign implementation are presented and compared.**

***Index Terms*—Embedded system, field programmable gate array (FPGA), fuzzy logic controller (FLC), genetic algorithm (GA), hardware/software codesign, intelligent vehicle.**

I. Introduction

Autonomous road-following is one of the major research topics in the area of intelligent vehicles [1], [2]. Generally speaking, autonomous road-following requires two major steps: road feature extraction and speed/steering control of the vehicles [3]. Control algorithms should be considered as an important issue in road-following to ensure safe and smooth rides. Human drivers can drive a car smoothly with their driving expertise rather than knowledge on control theory. This fact leads us to the fuzzy logic solution. Fuzzy logic control has been proven to be an effective and active method in solving control problems during the past decades. Fuzzy logic controllers (FLCs) provide a means of converting a linguistic control strategy based on expert knowledge into an automatic control strategy [4], [5].

FLC implementations can be divided into two categories: general-purpose-processor implementation with flexible fuzzy behaviors and dedicated hardware implementation for specific applications [6]. The general-purpose-processor implementation can be developed quickly and applied in various fields. It can process instructions adequately when the tasks are not too complicated. However, the general-purpose-processor implementation cannot guarantee real-time response when the complexity of tasks increases or multiple tasks need to be executed simultaneously. The hardware implementation of a FLC has better performance compared with the single processor. Even software implementations with multiple processors can hardly compete with hardware implementations in terms of execution speed. However, hardware implementations require longer development time and can only be used in restricted fields [7].

A preferable FCL hardware implementation should meet two requirements: flexibility and good performance. In the past few decades, reconfigurable hardware, which provides a compromise between specific-purpose hardware and general-purpose processors has received increasing attention in implementing algorithms due to its adaptability in diverse applications, real-time performance, and a short developing period. Field programmable gate arrays (FPGAs), an example of reconfigurable hardware, are flexible because they can be reconfigured and reused easily. By designing logic components on FPGAs, we can easily verify and turnaround the design by programming the configuration stream.

Hardware implementations usually have better real-time performance, but higher cost and longer development time compared with software implementations. Nowadays, hardware/software codesign, which aims to find the tradeoff between hardware, software, and scheduling schemes in terms of performance and cost optimization is common in embedded system design. Partitioning the system into hardware and software implementations is the major concern in hardware/software codesign [8], [9].

A lot of work has been done on hardware/software partitioning. Some of the existing partitioning approaches are: mathematical programming, greedy heuristics, and iterative improvement approaches. Huiqun and Wong [10] present an integrated hardware/software partitioning and scheduling strategy, where the partitioning process uses the information provided by the scheduling solutions as a guide. Ernst *et al.* [11] use a binary search algorithm to minimize the hardware price and satisfy other constraints. Mathematical programming can only be applied to small instances of the codesign problem. Search algorithms and iterative improvement algorithms suffer from the limitation that they may become trapped in local minima. Genetic algorithms (GAs) are known for success in searching solutions in a global manner. GAs are applied in hardware/software codesign in [13] and [14]. However, [13] only allows one processor, while the cost and execution time of communication is not considered. Dick and Jha [14] utilize the multi objective GA, which borrows the Pareto-optimal theory from the Game theory. However, this method does not consider the concurrency of two or more tasks. Moreover, because of the complexity of this partitioning method, the computation resource used for partitioning may raise the overall codesign expense.

In this paper, we will design an autonomous road-following embedded system for intelligent vehicles. A FLC will be developed to control the steering wheel of the vehicle. The proposed FLC is carried out on the FPGA board. With the help of electronic design automation (EDA) tools, the very high speed integrated circuits hardware description language (VHDL) program that models the system is directly used for synthesis, verification, and implementation [12]. Furthermore, a novel partitioning method using the GA is developed. The complexity of the proposed method is between the complexity of [13] and [14]. This method is capable of finding the tradeoff among several evaluation factors under conditions of constraints. It utilizes multiple chromosomes for each individual, so that concurrency of several tasks is acceptable for codesign. Moreover, it takes communications between various devices into consideration; therefore, the cost of the codesign solution can be predicted more precisely. The rest of the paper is organized as follows: the kinematic model of the vehicle and the experimental setup are studied in Section II. Section III presents the FPGA design of the FLC. Section IV represents the optimization of the system implementation using software/hardware codesign. Simulation and experimental results are shown in Section V. The last section of this paper concludes the research.

II. Experimental Setup and Modeling of the Car

The road-following problem of the intelligent vehicle with kinematic constraints in the 2-D workspace is studied. Considering the robotic vehicle modeled in Fig. 1, the rear wheels are aligned with the vehicle, while the front wheels are allowed to pivot about the axes. Let

Manuscript received September 8, 2009; revised January 26, 2010; accepted April 7, 2010. Date of publication June 3, 2010; date of current version October 15, 2010. This paper was recommended by Associate Editor X. Guan.

The authors are with the Department of Electrical and Computer Engineering, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: yi.fu@unb.ca; howard@unb.ca; kaye@unb.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCC.2010.2049262

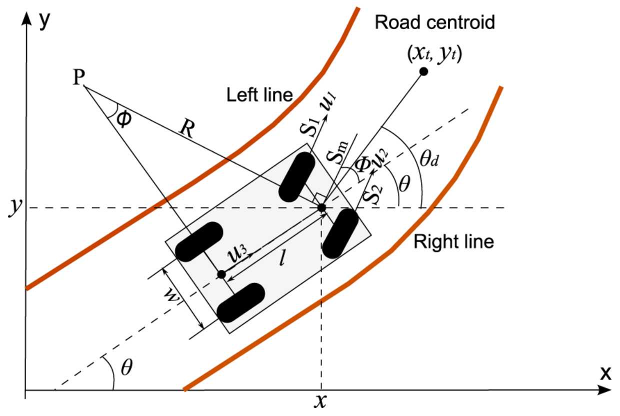

Fig. 1. Kinematic model of the car.

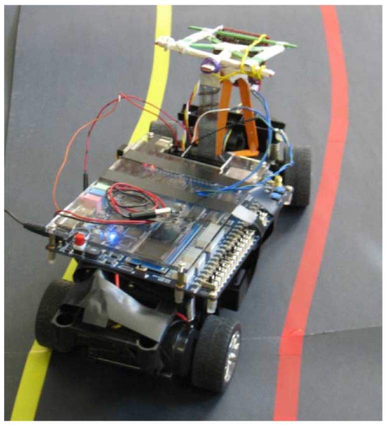

Fig. 2. Steering mechanism of the RC car.

$(x,\ y,\ \phi,\ \theta)$ denote the configuration of the robot, parameterized by the location of the front wheels. The kinematic model of the robotic vehicle can be represented as follows:

$$\begin{aligned}\dot{x} &= u_3 \cos\theta \\ \dot{y} &= u_3 \sin\theta \\ \dot{\theta} &= \frac{u_3}{l}\tan\phi\end{aligned} \tag{1}$$

where u_3 corresponds to the forward velocity of the vehicle and the angle of the vehicle body with respect to the horizontal line is θ, the steering angle with respect to the vehicle body is ϕ, $(x,\ y)$ is the location of the center point of the front wheels, and l is the length between the front and the rear wheels.

The experimental setup of the road-following system is shown in Fig. 2. The road-following system is implemented in a way that the vision-based vehicle can remain between two lines while simulating highway motion. The hardware system includes a camera with a built-in processor, a FLC implemented on the FPGA, and a radio control (RC) car with a fully functional steering and speeding system. This system utilizes the camera to find the center of the road. Then, the built-in processor of the camera calculates the error e, and sends it to the FLC through an RS-232 port. The error is proportional to the deviation presented in pixels. The FPGA decides how to turn the vehicle and generates pulse width modulation (PWM) signals according to the specification of the car to control the steering wheel. The speed and steering of the car are controlled by varying the PWM signals. The Altera DE2 board with a Cyclone II EP2C35F672C6 FPGA is used to interface with the camera and the car's existing circuitry, calculating the steering angle.

III. Design of the FLC

A block diagram of the specified FLC for the road-following task is shown in Fig. 3. The desired orientation of the center line of the car should be aligned with the road centroid. The error is the angle between the desired orientation of the center line and the actual center line of the car. The error is represented by $e = \theta_d - \theta$. To reduce the error to zero, the steering angle should be equal to ϕ. ϕ is determined by the FLC. The error and the change-in-error are calculated and fed into the FLC. The FLC is designed to output control signals corresponding to the control torque to the front steering motor to control the front wheels' steering angle ϕ.

When the car travels on the road, the DE2 board mounted on the car receives error data from the camera through the RS-232 serial port.

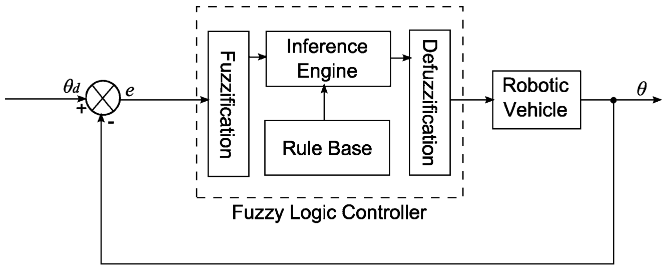

Fig. 3. Block diagram of the specified FLC.

The hardware architecture of the specific road-following circuit on the FPGA is shown in Fig. 4. In total, there are four I/O pins for this system: "clk" is the clock signal generated by the built-in 50 MHz quartz oscillator on the FPGA board. This signal is used to synchronize various modules of the system; "reset" is the input signal used to reset the system, "RX" is the signal sent from the camera through an RS-232 port, and the output port "CTRL" is an general purpose input/output (GPIO) pin, which sends PWM control signals to the car's circuitry that controls the steering wheel. The circuit of the proposed control system contains four main modules: 1) the universal asynchronous receiver/transmitter (UART) processing module, which processes the serial signals sent from the camera through the RS-232 port; 2) the change-in-error calculating module, which computes the change-in-error value based on the previous error and the current error; 3) the FLC module, the core circuitry of the design, which performs the fuzzy logic computation to determine the steering angle; and 4) the output processing module, which generates steering wheel control signals according to the calculation of the FLC.

The FLC module is the core circuitry of the road-following control system. It consists of a fuzzification submodule, a decision-making submodule, and a defuzzification submodule. The fuzzification submodule converts error and change-in-error crisp values to fuzzy values. The decision-making submodule includes the rule base and the inference engine, which make decisions about the steering angle according to fuzzy linguistic rules. Finally, the defuzzification submodule converts the resulting fuzzy values back to crisp values. The inputs of this module are 16 bits, where 8 bits represent the error and another 8 bits represent the change-in-error. The outputs are 8 bits, representing the crisp output value. The input and the output are in the 8-bit standard logic vector format. The implementation specifications of the proposed FLC are as follows:

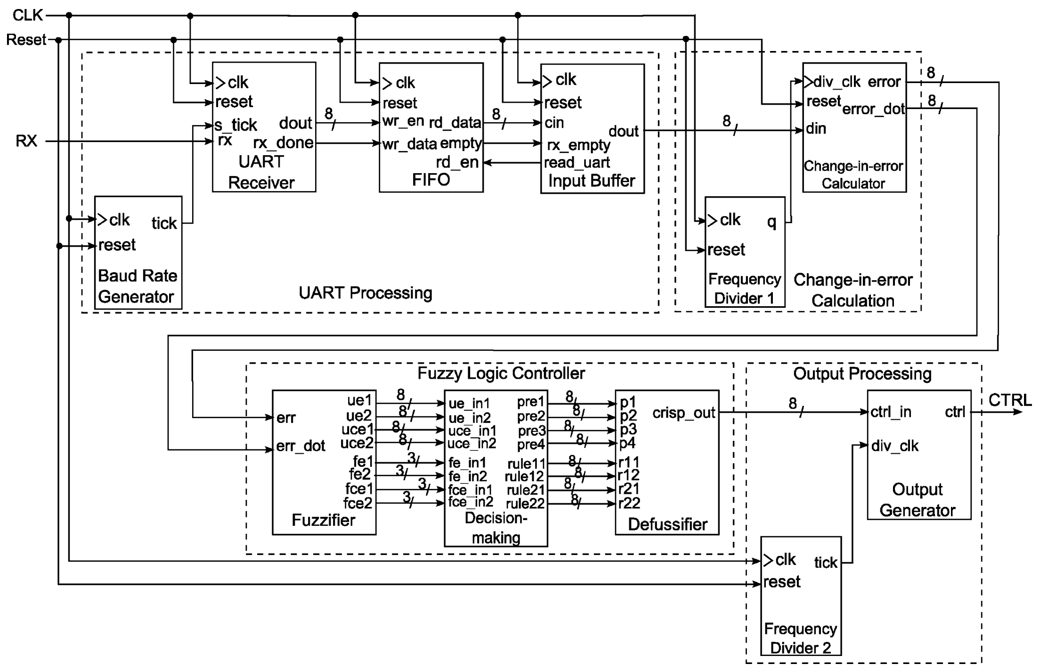

Fig. 4. Hardware configuration of the FLC.

1) number of input variables = 2 (error e and change-in-error $\dot{e}$);
2) range of input variables: [0, 100] (error and change-in-error are hexadecimal numbers; 8 bits are used for each input);
3) number of output variables = 1 (steering angle ϕ);
4) range of output variables: [0, 120] (output variables are hexadecimal numbers; 8 bits are used for the output);
5) shape of membership functions (MFs): isosceles triangles or trapezoids;
6) number of fuzzy sets per input variable = (5, 5) (5 for e and 5 for $\dot{e}$);
7) number of fuzzy sets per output variable = (5);
8) resolution of membership values = 8 bits (256 values for each membership value);
9) maximum overlapping between two MFs = 50%.

1) Fuzzification: This submodule judges which MFs are triggered and calculates the membership value of each triggered MF. The hardware architecture of this submodule is shown in Fig. 5. The inputs of this submodule are the 8-bit error vector and the 8-bit change-in-error vector from the change-in-error calculating module. There are 44 pins in total for the output port. "ue1" and "ue2" represent the membership values of the two triggered MFs for the error, and "fe1" and "fe2" are their corresponding fuzzy linguistic indexes. "uce1" and "uce2" represent the membership values of the two triggered MFs for the change-in-error, and "fce1" and "fce2" are their fuzzy linguistic indexes. Since there are five MFs for both inputs, we can use a 3-bit vector to indicate these four fuzzy linguistic indexes ($2^3 = 8 > 5$). The shapes of MFs, and the mapping between every input interval and a fuzzy set are predefined in the VHDL code. The number of mappings for each input variable is equal to the fuzzy set number (i.e., 5). Each input variable is evaluated and mapped to a fuzzy set. Then, according to the value of this variable, a membership value is assigned. We use two indexes for each input to indicate which two MFs are triggered.

2) Decision-Making Submodule: Fig. 6 depicts the architecture of this submodule. It contains the fuzzy rule base, the inference engine, and the antecedent fitness block. The decision-making submodule takes

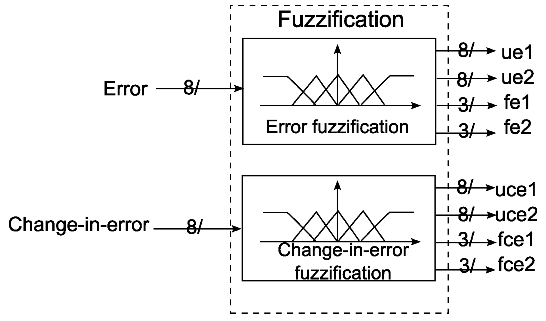

Fig. 5. Fuzzification submodule.

the the membership values and the fuzzy linguistic indexes exported from the fuzzification submodule as inputs. According to the input membership values and indexes, it makes decisions based on a predefined fuzzy rule base.

Since the "AND" fuzzy logic operator is used for getting antecedent values, "Min" blocks shown in Fig. 6 are implemented to select the minimum membership values. Four membership values exported from the fuzzification submodule ue1, ue2, uce1, and uce2 are compared and selected with four 2-to-1 "Min" circuits in order to get the antecedent values of the fuzzy rules.

Four fuzzy linguistic indexes from the fuzzification submodule are grouped into four combinations each of which contains one index from each of the inputs. Each combination is used to choose a control rule according to the predefined rule base. The linguistic fuzzy indexes and the selected rules are associated with IF–THEN fuzzy conditional statements. In VHDL, this association is defined as IF–THEN branch statements, which can be expressed as follows:

IF $(fe1 = PosL$ AND $fce1 = PosL)$
THEN $R := RgtL;$
ELSE IF $(fe1 = Zero$ AND $fce2 = Zero)$

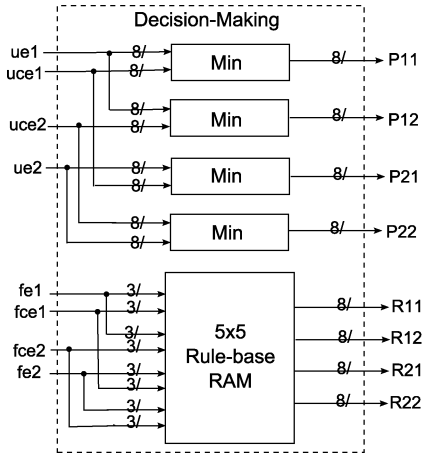

Fig. 6. Decision-making submodule.

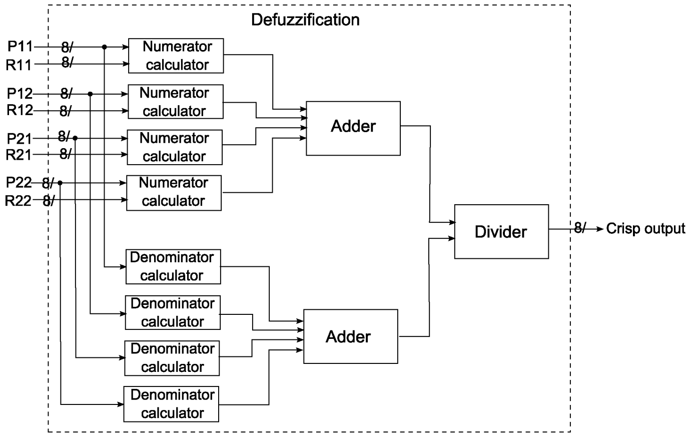

Fig. 7. Defuzzification submodule.

THEN $R := Mid$;
ELSE IF ($fe2 = NegS$ AND $fce1 = PosL$)
THEN $R := RgtS$;
ELSE IF ($fe2 = PosS$ AND $fce2 = PosS$)
THEN $R := RgtL$;
$\ldots$;

Although only four rules are listed earlier for illustration, there are 25 fuzzy control rules in the rule base.

3) Defuzzification Submodule: The defuzzification method is based on the center of gravity method mentioned in the previous section. The configuration of this submodule is illustrated in Fig. 7. The four triggered rules and their corresponding antecedent values exported from the decision-making submodule are used as the input signals of this submodule.

IV. Optimizing the Road-Following System With Hardware/Software Codesign

To maintain the real-time performance of embedded systems while keeping the cost low, hardware/software codesign is usually applied during embedded system implementation. In this section, a novel hardware/software partitioning approach is proposed to optimize the implementation of the road-following system designed in previous sections.

GAs mimic the processes of natural genetic systems to solve optimization problems by encoding the possible solutions into chromosome-like strings of data [15]. Similar to other search algorithms, GAs perform a search on a multidimensional space containing a hypersurface known as the fitness surface. This ability makes GAs excel at finding the tradeoff among several factors.

A novel hardware/software partitioning method using the GA is developed. This method is capable of finding the tradeoff among several evaluation factors under conditions of hard constraints. A small-scaled intelligent vehicle, which is capable of autonomous road-following is built and the proposed controller is tested in the real-world environment.

A. Proposed Method

In this partitioning problem, each individual is represented as chromosomes, each of which contains many genes. The number of chromosomes one individual has equals to the number of task flow paths and each chromosome represents one task flow path. The number of genome segments in one chromosome equals to the number of tasks in the corresponding task flow path.

In order to find the near-optimal solution, some specific device assignments should be found. As the number of tasks and available devices increase, it is quite time-consuming to search a specific solution in a big search space. Each point in the search space represents one feasible solution. For the partitioning problem, all the possible combinations of device assignments for tasks are solutions in the search space. Each feasible solution can be marked by its fitness for the problem. Finding the solution is then equal to looking for some extreme value in the search space that leads to the optimal fitness. An outline of the GA optimization procedure for the proposed partitioning method is listed as follows:

1) define the population size;
2) initialize generation 0 of the population. Encode the device assignment for each task to get their chromosomes;
3) evaluate each partitioning solution in the population;
    a) if a satisfactory solution is found, go to step 5;
    b) if none of the solutions in this generation meets the requirement, go to step 4;
4) reproduce the new generation;
    a) sort by the fitness value;
    b) select good individuals;
    c) crossover;
    d) mutate and get new offsprings, go to step 3 with the new generation;
5) store the near-optimal partitioning solution selected by the GA and use it as the final solution.

B. Fitness Function

The chromosomes are evaluated according to a fitness function. It is important to select a good fitness function, otherwise the most feasible chromosomes may be eliminated because of the noise caused by poor fitness evaluation.

The fitness function is modeled as follows:

$$f_m = \frac{1}{\Sigma_{j=1}^{5} a_j F_j} \tag{2}$$

where f_m is the fitness function for one solution, m is the solution index number, and j is the evaluation criterion index. Five evaluation criteria are considered: F_1 is the total performance violation of the partitioning solution in terms of missed deadline, F_2 is the violation in terms of concurrency, F_3 is the total execution time of the partitioning implementation, F_4 is the total price of the implementation, and F_5 is the total power consumption. a_j is the weight factor for the jth evaluation criterion.

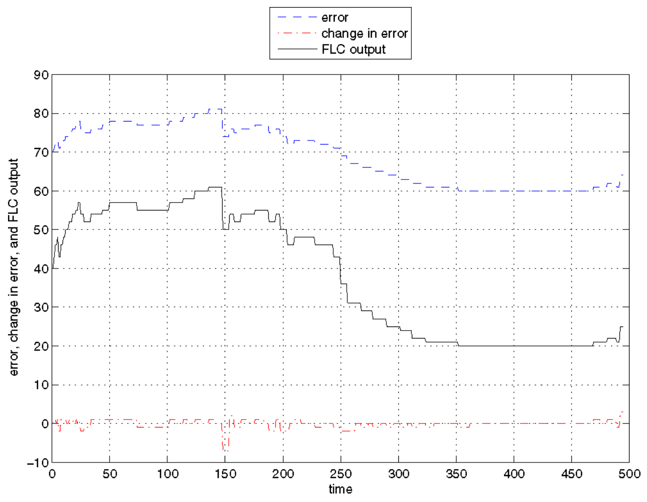

Fig. 8. e, $\dot{e}$, and Φ during the experiment.

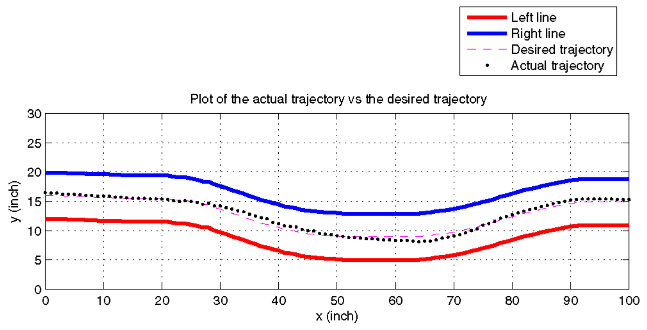

Fig. 9. Actual and the desired trajectory.

V. Experiment Results

In order to verify the performance of the proposed methods, experiments have been conducted.

A. Experiment Results of the FPGA Implementation

After simulation, synthesis, and compilation, the configuration stream is programmed into the FPGA board for road-following tests. We use a mocked road with two lines to guide the navigation of the intelligent vehicle. The car is able to follow the road at a speed of 0.9 m/s. Quartus's SignalTap II Logic Analyzer is used to record experiment data, while the car follows the curved road. Error (e), change-in-error ($\dot{e}$), and the FLC control output (Φ) are recorded and plotted in Fig. 8. Fig. 8 shows the actual control output of the FLC with respect to inputs. It matches the simulation result. The actual trajectory of the car is recorded and compared with the desired trajectory, which is the center line of the road in Fig. 9, where the actual trajectory is the average of five experiment data. The maximum absolute error between the desired trajectory and the actual trajectory of the intelligent vehicle is 1.02 in. The root-mean-square error is 0.56 in, which is relatively small compared with the width of the road that is 8 in.

B. Experiment Results of the Partitioning

The goal of the partitioning is to move some tasks from hardware to software, so that the cost of the embedded system will be reduced while the specification is met. The task flowchart of the road-following system is generated first. Then, the performance and cost estimation

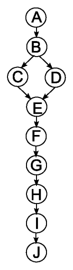

Fig. 10. Task flowchart of the autonomous road-following system.

of tasks on different devices are evaluated according to the hardware and software implementation. The GA is used to find the partitioning solution. After a satisfactory partitioning solution is found, it is implemented on the Altera DE2 FPGA board, which has a Nios II processor. The hardware part is implemented on the FPGA, while the software part is implemented on the Nios II processor.

1) Task Flowchart: The functionality of the proposed road-following system is decomposed into tasks of some granularity first. The level of granularity must be chosen carefully. If the granularity level is too deep, the performance and expense may be degraded because of extra communications, and the computation resource used for partitioning will be increased. However, if the granularity level is not deep enough, the satisfactory codesign solution may not be found.

The task flowchart of the road-following system is shown in Fig. 10. The functionality of the system is broken down into 10 tasks: $Task\ A$: reading errors; $Task\ B$: calculating change-in-errors; $Task\ C$: fuzzification for error; $Task\ D$: fuzzification for change-in-error; $Task\ E$: selecting fuzzy logic rules; $Task\ F$: computing the antecedent values; $Task\ G$ through $Task\ I$: calculating the crisp output; $Task\ J$: generating PWM signals to control the car. We define $Task\ C$ and $Task\ D$ as concurrent tasks. They must be executed simultaneously.

2) Evaluation of Hardware/Software Implementation: The time constraint for $Task\ A$ is defined based on the baud rate for the RS-232 transmission. To meet the specification of the circuit on the car, $Task\ H$ must be finished within 25 ms and to guarantee the reliability of the system, all tasks must be finished within 24 ms. Although all tasks must be finished within 24 ms, there is no hard time constraint for each individual task. Hence, we can define flexible time constraints within reasonable ranges for tasks, but a solid time constraint for the whole functionality. The time constraints are shown in Table I.

Table II shows the cost of each task on each device as well as the communication cost. The price of hardware is evaluated by the number of logic elements used to implement the user-defined logic. We define the price of software implementation to be 0 because the software already exists and no extra cost will be added if some tasks are executed on software. The communication is considered if two consecutive tasks are executed on different devices. In this case, communication can happen on outputs of $Task\ A$ to $Task\ I$, which are logic vectors. The parallel input/output (PIO) is used for communication between the FPGA and the Nios II processor. The PIO module is a library component included in the Nios development kit, providing a memory-mapped interface between the software module and the hardware module. The communication time with PIO modules can be neglected compared with other tasks executed on the software. Small values are used for communication price and power consumption of each PIO. The communication price and power consumption between tasks are estimated based on the number of PIO used.

The partitioning result using the proposed GA is shown in Fig. 11. The population in each generation is 50 and 50 generations are pro-

TABLE I
TIME CONSTRAINTS OF TASKS

$Task$	A	B	C	D	E	F	G	H	I	J
$Time\ Constraint\ (ms)$	2	0.5	0.25	0.25	0.5	0.5	0.5	0.25	0.25	19

TABLE II
COST OF TASKS AND COMMUNICATION

$Device$	Cost	Task A	B	C	D	E	F	G	H	I	J
$FPGA$	Exec. Time (ms)	1	0.1	0.05	0.05	0.1	0.1	0.1	0.05	0.05	18
	Price	971	873	607	607	515	479	981	497	598	938
	Power	20	18	12	12	10	10	25	10	10	18
$Nios\ II$	Exec. Time (ms)	185	0.5	0.2	0.2	0.3	0.5	0.3	0.2	0.2	18
	Price	0	0	0	0	0	0	0	0	0	0
	Power	18	14	10	10	8	8	14	8	10	16
$Communication$	Comm. Time (ms)	/	0	0	0	0	0	0	0	0	0
	Price	/	5	10	5	5	30	40	40	10	5
	Power	/	0.1	0.2	0.1	0.1	0.6	0.8	0.8	0.2	0.1

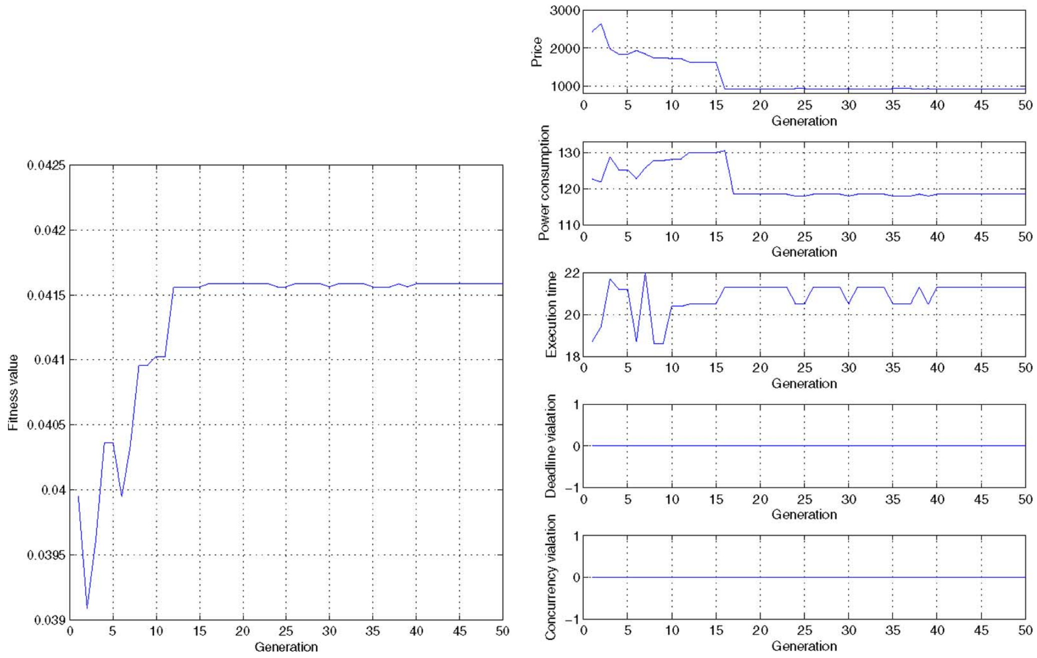

Fig. 11. Fitness values, costs, and violations.

duced to get the solution. The time constraint and the price term are weighted heavily to avoid deadline violation and high implementation price. The final partitioning solution is to execute $Task\ A$ on the hardware, while to execute other tasks on the software. Thus, the hardware module receives data through the RS-232 port from the camera and converts the data into standard logic vectors. The software module makes decisions on the steering control using the FLC and generates PWM signals. The hardware module is a user-defined logic. The software module is implemented on the Micrium's MicroC/OS-II operation system with the Nios II processor using C language. The configuration of the hardware/software codesign implementation is shown in Fig. 12.

With the codesign implementation, all tasks can be finished within 22 ms, which meets the real-time specification. The number of logic elements required for the codesign is 971 compared with 7066 for the hardware design. Experiments have been conducted using the autonomous road-following robot. The robot is made to perform the autonomous road-following task. The robot is able to follow the road at a speed of 0.9 m/s. Experiment results of the codesign implementation is similar to the hardware implementation, but the cost is reduced significantly.

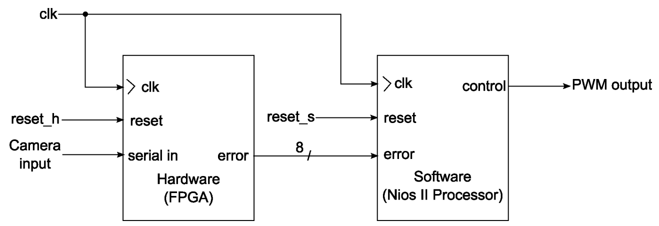

Fig. 12. Configuration of the codesign implementation on the DE2 board.

VI. CONCLUSION

In this paper, an intelligent vehicle that is capable of autonomous road-following is designed and implemented on the FPGA. A FLC implemented on the FPGA is utilized to control the steering wheel according to the deviation from the roads. Furthermore, a hardware/software partitioning method based on GA is designed for embedded systems. It utilizes multiple chromosomes for each individual solution to allow concurrency of multiple tasks. Experiments with both FPGA and hardware/software codesign implementation demonstrate that the vehicle with these two kinds of application can automatically follow the curved road. However, the codesign strategy leads to lower cost.

References

[1] W. Tsui, M. S. Masmoudi, F. Karray, I. Song, and M. Masmoudi, "Soft-computing-based embedded design of an intelligent wall/lane-following vehicle," *IEEE/ASME Trans. Mechatronics*, vol. 13, no. 1, pp. 125–135, Feb. 2008.

[2] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker, "A predictive controller for autonomous vehicle path tracking," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 1, pp. 92–102, Mar. 2009.

[3] Y. Fu, H. Li, and M. Kaye, "Design and stability analysis of a fuzzy controller for autonomous road following," in *Proc. IEEE Intell. Veh. Symp.*, Xi'an, China, Jun. 2009, pp. 66–71.

[4] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller. Part I," *IEEE Trans. Syst., Man Cybern.*, vol. 20, no. 2, pp. 404–418, Dec. 1990.

[5] T. H. S. Li, S. Chang, and Y. Chen, "Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot," *IEEE Trans. Ind. Electron.*, vol. 50, no. 5, pp. 867–880, Oct. 2003.

[6] D. Kim, "An implementation of fuzzy logic controller on the reconfigurable FPGA system," *IEEE Trans. Ind. Electron.*, vol. 47, no. 3, pp. 703–715, Jun. 2000.

[7] C.-F. Juang and Y.-W. Tsao, "A type-2 self-organizing neural fuzzy system and its FPGA implementation," *IEEE Trans. Syst., Man, Cybern.*, vol. 38, no. 6, pp. 1537–1548, Dec. 2008.

[8] R. Niemann, *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems*. New York: Springer-Verlag, 1998.

[9] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Hardware/software codesign: A systematic approach targeting data-intensive applications," *IEEE Signal Process. Mag.*, vol. 22, no. 3, pp. 14–22, May 2005.

[10] L. Huiqun and D. F. Wong, "Integrated partitioning and scheduling for hardware/software co-design," in *Proc. Int. Conf. Comput. Des.: VLSI Comput. Processors*, Oct. 1998, pp. 609–614.

[11] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Des. Test Comput.*, vol. 10, no. 4, pp. 64–75, Dec. 1993.

[12] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. New York: Wiley-Interscience, 2008.

[13] D. Saha, R. S. Mitra, and A. Basu, "Hardware software partitioning using genetic algorithm," in *Proc. Tenth Int. Conf. VLSI Des.*, Jan. 1997, pp. 115–1603.

[14] R. P. Dick and N. K. Jha, "MOGAC: A multi objective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 10, pp. 920–935, Oct. 1998.

[15] L. Davis, *Handbook on Genetic Algorithms*. New York: Van Nostrand/Reinhold, 1991.