Current Development in Artificial Intelligence.
ISSN 0976-5832 Volume 2, Number 1 (2011), pp. 1-9

http://www.irphouse.com

Real Time Implementation using Partial Reconfiguration

S.S. Shriramwar[1] and N.K. Choudhari[2]

[1]Priyadarshini College of Engineering, Nagpur University, India
E-mail: sshriramwar@yahoo.com
[2]Smt. Bhagwati Chaturvedi College of Engineering, Nagpur University, India
E-mail: drnitinchoudhari@gmail.com

Abstract

Dynamic Partial Reconfiguration (DPR) of FPGAs presents many opportunities for application design flexibility, enabling tasks to dynamically swap in and out of the FPGA without entire system interruption. In this paper, a line follower robot for the white line as well as for black line was implemented on Virtex-2 FPGA board. These modules were dynamically reconfigured on Virtex-2 FPGA board in the run-time. This design includes two modules one is static and the other is partially reconfigurable regions (PRR) which is a dynamic region. The controllers were the static modules used for controlling the flow of data to and from the reconfigurable modules to the external world (host environment) through bus macros, whereas white line and black line modules were designed as dynamic modules. Different hardware modules were used such as sensors and actuators, all these modules were interfaced with FPGA controller. The speed of motor was controlled using pulse width modulation (PWM) using VHDL.

Keywords: Partial Reconfiguration, Field Programmable Gate Array (FPGA), mobile robot, actuators, Sensor.

Introduction

The emergence of reconfigurable Field Programmable Gate Arrays (FPGA) has given rise to a new platform of complete mobile robot control system. With FPGA devices, it is possible to tailor the design to fit the requirements of applications (for example, exploration and navigation functions for a robot). General-purpose computers can provide acceptable performance when tasks are not too complex. A single processor system cannot guarantee real-time response (particularly in the absence of

considerable additional hardware), if the environment is dynamic or semi-dynamic. An FPGA-based robotic system can be designed to handle tasks in parallel.

Normally a mobile robot system includes several tasks, such as path planning, obstacle avoidance, emergency stop, sensor signal processing, sensor manager for sensor fusion, actuators, and actuator manager. Most robot systems are implementing all these tasks in software. It would be a challenging job for a software engineer to implement all the tasks under real-time constraints especially with computation-intensive image information under dynamic environment. The expensive ASIC can fulfill the speed criteria; however, it is a complicated and expensive procedure if a slight change occurs. Recent attempts to find new ways to speed up computation take advantage of the FPGA, since they offer the speed similar to an ASIC with a flexibility equal to or even higher than a general processor. In order to increase the robot's robustness and flexibility with the real-time constraints, in this paper, we propose a dynamically reconfigurable real-time embedded system platform by integrating high-performance microprocessors with a reconfigurable FPGA to enable a mobile robot system to adapt to dynamic environment on the fly with the minimum loss of its performance and capabilities.

Recent evolution in FPGA technology allows the designer to update/reconfigure only a specific part of the internal structure of the FPGA at run-time using a technique known as Partial Dynamic Reconfiguration (PDR). With PDR, it is possible to implement dynamic systems that adapt to run-time needs of the application allowing reuse and reconfiguration of hardware cores (IP blocks). This allows the FPGA to remain operational without comprising the integrity of the applications running on parts of the FPGA that do not undergo reconfiguration.

Design overview

Architecture of mobile robot

The top-level architecture of the prototype is shown in figure 1. It supports ADC to sense the distance of the obstacle from vehicle with the help of GP2D12 sensor and LCD to display distance of the vehicle from `obstacle and` relative speed of the vehicle. Within the proposal of mobile robotics platform, the use of FPGA Controller, with control software especially developed for the necessary applications is considered using structured libraries to design, simulation, and verification with SIMULINK, we convert the model to function prototyping using FPGA hardware.

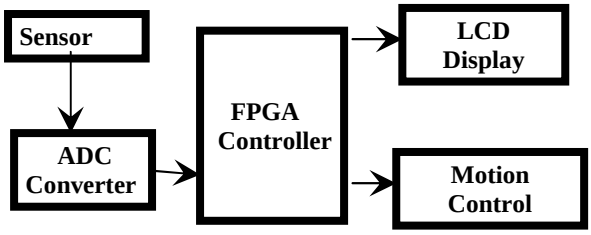

Figure 1: Top level architecture of prototype.

Interfacing the FPGA with ADC 0809

An A/D converter translates an analog signal into a digital value. An 8 channel, 8-bit A/D input is available to read analog voltages between 0 to 5 Volts. Devices such as an analog joystick or potentiometers can be connected to one of ADC channel and converted digital output can be read and is sent back to the FPGA board to control the speed of DC motor. The characteristics of an A/D converter include:

- Accuracy: expressed in the number of digits it produces per value
- Speed: expressed in maximum conversions per second (for example 500 conversions per second)
- Measurement range: expressed in volts

Interfacing the FPGA with L293D

The FPGA will process the PWM program and the output will be given to enable pin of L293D which activate the L293 quadruple high current half H-Driver chip and controls the speed of the motor.

LCD module

Recently, a number of projects use intelligent liquid crystal display (lcd) modules. There ability to display not just numbers, but also letters, words and all type of symbols makes them a good deal more versatile than the familiar 7-segment light emitting diode (led) displays. The LCD module used in this design is a 2 line 16 characters per line FPGA module. This LCD module is available on CPLD board itself.

Distance measurement sensor

The analog sensor Sharp GP2D12 simply returns a voltage level in relation to the measured distance. In Figure 2, above, the relationship between digital sensor read-out (raw data) and actual distance information can be seen. From this diagram it is clear that the sensor does not return a value linear or proportional to the actual distance, so some post-processing of the raw sensor value is necessary. The simplest way of solving this problem is to use a lookup table which can be calibrated for each individual sensor.

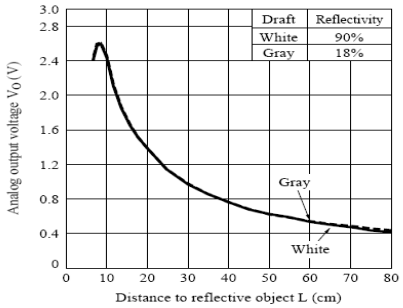

Figure 2: Analog output voltage vs distance to reflective objects.

Functional description

A GP2D12 distance sensor, which is mounted on the vehicle model senses the obstacle in the range of 10 cm to 80 cm and generate an analog output voltage as shown in figure 3, which is nonlinear with respect to the distance. Hence a proper calibration method is required. ADC 0809 then converts this analog voltage into 8 bit digital data. As maximum output voltage from sensor is 2.55V, we set the Vref of ADC to 2.55V. On the ADC0831 this will give a value of 0 to 255 for an input voltage of 0 to 2.55 volts. This gives us a resolution of 0.01 volts per step from the ADC. FPGA senses the 8-bit output of ADC and compare it with the data present in the synthesized registers of FPGA for a fixed distance interval. When sensed data matches with data in FPGA, FPGA transfers ASCII code for respective distance to be displayed on LCD module. Thus LCD displays the distance of vehicle model from obstacle in a real time continuously. A PWM signal is generated by inferring hardware in FPGA based on the output from ADC. When vehicle model is at 80 cm from obstacle, this signal has more ON time period. Hence DC motor in vehicle model gets more average power. We decoded ADC output such that the speed of vehicle model will be 7 cm/sec at 80 cm distance and then it will decrease by 1 cm/sec for decrease in distance by 10 cm and so on. Thus at 10 cm distance of vehicle model from obstacle, the speed will be 0 cm/sec i.e. vehicle will stop. Thus vehicle would not hit the obstacle. As this PWM signal generated by FPGA cannot drive the DC motor of vehicle model, a L293D motor driver IC is used. A PWM signal is applied to the enable pin of driver IC and DC motor of vehicle model is connected to output1 and output2 pins of driver IC. A 270 KHZ clock is provided to ADC as well as FPGA because of its availability on FPGA board. A higher clock can be used to improve the speed of operation of the design, as ADC0809 can work up to 1280 KHZ and FPGA can even work on MHz range.

Synthesis results

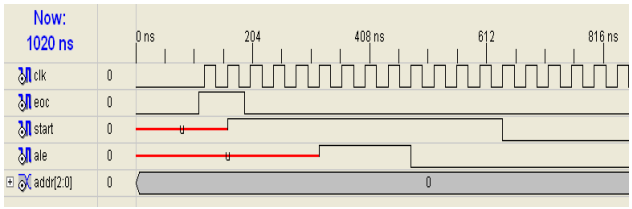

Figure 3: Simulation result of ADC interface.

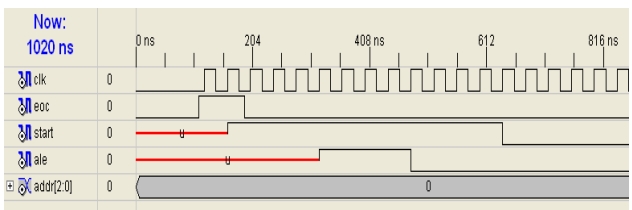

Figure 4: Simulation result of motor interface.

The Figure 5 shows the snap Shot of Mobile Robot Prototype. After the simulation and the synthesis process, the program has been implemented on the FPGA board.

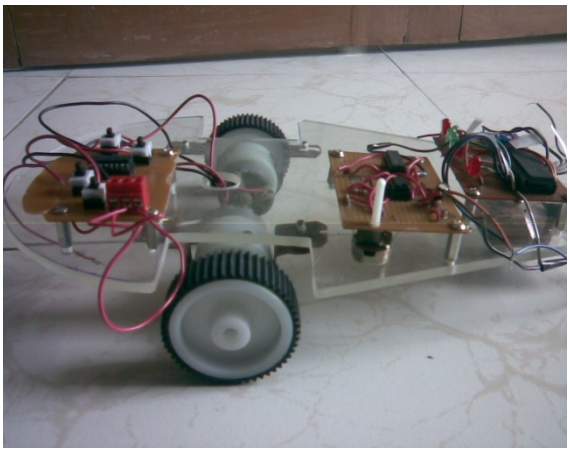

Figure 5: A snap Shot of Mobile Robot Prototype.

Partial reconfiguration

Partial Dynamic Reconfiguration is the capability to reconfigure specific areas of the FPGA at run-time after its initial configuration. PDR is carried out to allow the FPGA to adapt to changing hardware algorithms, improve fault tolerance and resource utilization, to enhance performance or to reduce power consumption.Xilinx proposed the initial PDR methodology in [8] and currently only Xilinx FPGAs support Partial Dynamic Reconfiguration. Initial Xilinx PDR design flows recommended the use of tri-state buffers for communication between dynamic and static parts of the FPGA [8]. They were not very effective leading to a new alternative in the form of predefined routed macros generally termed as bus macros [9]. Bus macros are placed at the edge of boundaries separating dynamic and/or static regions/modules. They allow communication between the different regions of the FPGA. The limitation to this approach was that although signals could pass through the reconfigurable area by use of bus macros, however if a module was being reconfigured, the internal signal route was also re-routed making the parts of the systems on either side of the module isolated from each other. This problem was addressed later on. Virtex devices support glitchless dynamic reconfiguration: If a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation, with the exception of LUT RAM and SRL16 primitives in Virtex-II/Pro devices. This limitation has been removed in the Virtex-4 family.

Implementing designed prototype using partial dynamic reconfiguration

Conceptually, the reconfigurable fabric of the system is divided in two areas, the static area and the dynamically reconfigurable area. Hardware modules that remain unchanged during the whole application execution are placed in the static area. The dynamic area is time-shared between the dynamic hardware modules; they are loaded as needed by the application running on the embedded CPU with the help of the run-time management system. The hardware platform used for this work is a board with a Xilinx XC2VP7 FPGA and 32 MB of external memory. The FPGA contains an embedded PowerPC (PPC) 405 processor. The static area of the system contains the following modules (more can be added if necessary):

- A memory interface unit; the external memory stores both the reconfiguration data of the dynamic modules and the application-specific data.
- A reconfiguration control unit that performs the reconfigurations using the Virtex-II Pro Internal Configuration Access Port (ICAP).
- Two I/O units: one to communicate with the modules in the dynamic area and one to transfer data To/from external devices (for instance, a controlling computer).

As can be seen in the figure.6, the system is connected to an external Flash memory (which stores the partial bit streams to carry out the reconfiguration) via a Flash controller which provides read/write access to the external memory. The ICAP module is present to read/write configuration data to/from the devices' configuration memory. An internal on-chip Block RAM can be utilized to store the recently read configuration data (not shown in the figure). Reconfiguration of the dynamic region is managed by a Reconfiguration Controller which can be either external or internal (self configuration) to the FPGA. It is responsible for the partial reconfiguration bitstream that must be loaded onto the reconfigurable modular region. However, self configuration can be carried out through the internal ICAP parallel interface and reduces reconfiguration time as compared to external reconfiguration. For this reason, our reconfiguration controller is present inside the FPGA and communicates to the rest of the system via the Processor Local Bus (PLB). Other peripherals are attached to a slower On-Chip Peripheral Bus (OPB) via the PLB-OPB Bridge.

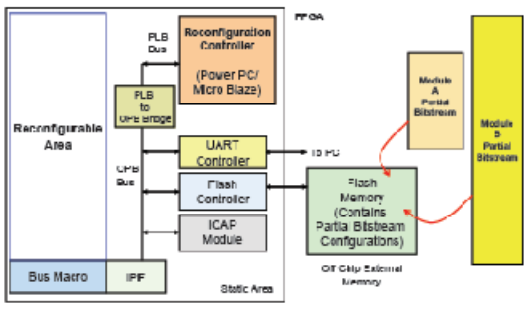

Figure 6: Conceptual model of complete system to carry out PDR.

Algorithm for Implementation

In this section we describe the algorithm for implementing the prototype on Virtex-II Pro FPGA using Hardware Reconfiguration. The first step is to specify the base system with help of the Xilinx EDK. VHDL descriptions of the system and the synthesized peripherals are then exported to the Xilinx Integrated Synthesis Environment (ISE) [7] for implementation. The results of this step may be reused for multiple projects;. The next step is to add the static area part of the Bus Macro to the design, in the form of a relatively placed and synthesized design, and connect it to the inputs/outputs of the OPB Dock. The creation of this Bus Macro design is made in a different ISE project. After translation, mapping and routing of the main design, we obtain the NCD file that defines the basis of the system (NDC is an internal format used by ISE).

After all designs have been assembled, we generate the partial and full reconfiguration files. The full reconfiguration file is generated from the base system design. The process of generating the partial configuration files depends on the layout of the modules in the dynamic area. For modules that occupy whole columns, a partial file is generated for each assembled design by the Xilinx bitstream generation tool (bitgen) or by ModBitCreator, a JBits [7] program we developed. The base system design is used as reference, so that the partial bitstreams created contain only the data for the modified frames (the basic unit of reconfiguration). The ModBit Creator also creates a partial bitstream that sets the range of columns spanned by the module to its base state. For modules that share columns, a partial file is generated per assembled design (having the base system design as the reference) by ModBit Creator. The partial bitstream created by this program contains the configuration data for all the frames that cover the range of columns occupied by the module. All partial reconfiguration files have to be transformed into Module-based Partial Reconfiguration (MPR) files, which are used by our run-time system to manage module information .These files are created by MPRCreator, an application we developed for this purpose. In order to be downloaded to the board, all MPR files are processed by the mfsgen tool , to create a Xilinx Memory File System (MFS) I mage.

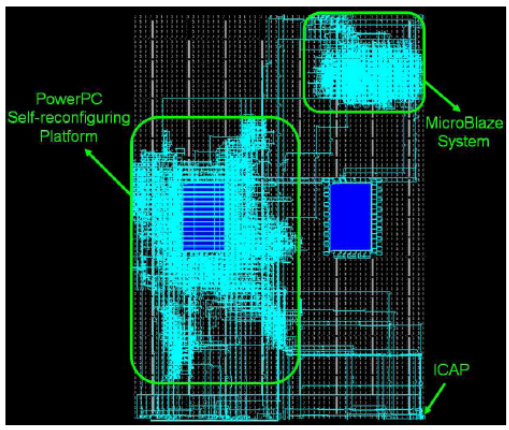

Figure 7: FPGA editor view of the difference-based design.

Implementation

Initial budgeting using UCF constraint file, an ASCII file specifying constraints on the logical design, three area groups were created in accordance with the guidelines in pervious section.

1. AREA GROUP "AG U1" RANGE = SLICE X0Y159 : SLICE X47Y0
2. AREA GROUP "AG U2" RANGE = SLICE X48Y159 : SLICE X83Y0
3. AREA GROUP "AG U3" RANGE = SLICE X84Y159 : SLICE X91Y0

Instances of the static, reconfigurable, and ICAP-JTAG wrapper modules were assigned to the above area groups respectively. TBUFs, BRAMs, and MULTs were also included in the ranges of the area groups. Also, all system level, FPGA pins, and bus macro location constraints were assigned. Active Module Implementation Module specific constraints such as DCM and PPC405 location constraints were added to the constraint file created in the previous phase.

To implement each module, the top-level and module netlists in NGC format and the constraint file were used. Each active module was separately expanded into the top-level design. It then mapped, placed, and routed. The implemented designs were published as Physically Implemented Module (PIM) to a different directory to be used in the next phase.

Final Assembly: This phase constructed a complete design using the top-level design files created during the initial budgeting phase, the top-level constraints file, and the implemented modules published to the directory in the previous phase. The final bitstream for the complete design was generated in this phase after running the following commands:

1. ngdbuild -p xc2vp30®896-6 -uc <constraint file>.ucf -u -modular assemble -pimpath ...n...nPims <top-level file>.ngc
2. map -pr b <top-level file>.ngd -o <top-level file> map.ncd <top-level file>.pcf
3. par -w <top-level file> map.ncd <top-level file>.ncd <top-level file>.pcf

Conclusions & Future work

In this paper we designed a prototype for obstacle detection application and successfully realized a method for performing secure partial reconfiguration of FPGAs by implementing a special this application using embedded processor cores. Under software control and within a single FPGA, the configuration controller was able to partially reconfigure an application system on the FPGA while the rest of device maintained correct operation. By performing partial bitstream encryption and authentication, this method improves the design security specifically for designs that benefit from partial reconfiguration. It also provides the flexibility of using arbitrary algorithms for authentication and encryption/decryption of partial bitstreams.

Among the next steps of this development work are:

i. Designing other modules.
ii. Providing run-time support for applications running on a controlling computer.
iii. Implementing these modules on Virtex-IV FPGA

References

[1] István Matijevics: Microcontrollers, Actuators and Sensors in Mobile Robots, in Proceedings of 4th Serbian-Hungarian Joint Symposium on Intelligent Systems (SISY 206) September 29-30, 2006, Subotica, Serbia.

[2] Volnie A.Pedroni. “Circuit Design with VHDL” .MIT Press, Cambridge, Massachusetts, London , England.

[3] Prabhas Chongstitvatana. “A FPGA-based Behavioral Control System for a Mobile Robot”. IEEE Asia-Pacific Conference on Circuits and Systems, Thailand, 1998.

[4] Steve Guccione, D. Levi, and P. Sundararajan. JBits: Java based interface for reconfigurable computing. In MAPLD’99, Maryland, September 1999.

[5] K. Sridharan and P. Rajesh Kumar. “Design and Development of an FPGA based Robot”. www.springerlink.com. Berlin Heidelberg 2008 .

[6] Xilinx. Embedded Development Kit Documentation, 2004.

[7] “Two Flows for Partial Reconfiguration: Module Based or difference Based,” in Xilinx Application Note XAPP290, Version 1.1, 2003.

[8] “Two Flows for Partial Reconfiguration: Module Based or Difference Based,” in Xilinx Application Note XAPP290, Version 1.2, Sept. 2004. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf

[9] B. Blodget and C. Bobda and M. Huebner and A. Niyonkuru, “Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs,” in FPL’04, 2004, pp. 801–810.

[10] P. Lysaght and B. Blodget and J. Mason and J. Young and B. Bridgford, “Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs,” in FPL’06, Madrid, Spain, Aug. 2006.