A reconfigurable multi-core computing platform for robotics and e-health applications

Dennis Majoe[2], Lars Widmer[1], Liu Ling[1], Jim Chih-Chen Kao[1], Jürg Gutknecht[1]
[1]*ETH Zürich, Switzerland*
Lars.Widmer@inf.ethz.ch
[2] *MA Systems and Control Ltd., United Kingdom.*
dennis.majoe@masystems.co.uk

Abstract

Increasingly embedded devices are turning to two technologies to achieve high performance and enable efficient programmability as well as product usability. The first is multi-core processing on FPGA devices in which the multi-core architecture allows software to map application-level parallelism to inherent parallel fabric to offer better performance, the re-configurability leads to flexible and adaptive designs. The second is wireless communications that allow sensors to be distributed flexibly across a structure for example in the case of a body area network. This paper describes the ongoing design of a multi RF channel, multi-core embedded design which will be used as a generic FPGA solution to meet the requirements of both e-health applications as well as robotics applications.

Keywords: FPGA, e-Health, robotics, embedded systems, reconfigurable multi-core processing, wireless communications.

1. Introduction

Embedded applications are becoming more complex requiring higher levels of parallelization and computational complexity. However as they often run on batteries they must be low power and additionally they must be small enough to fit into highly portable or wearable solutions. In this paper the embedded applications focus on robotics and e-Health sensors.

In the field of robotics an important aspect is re-configurability both during development for design and debug and post development for functional upgrades. As a reconfigurable device, the field programmable gate array (FPGA) introduces not only the software programmability commonly seen in microcontrollers, but also hardware programmability, that is, the hardware architecture can be fine tailored to the applications.

The early use of FPGAs in robotics aimed to achieve highly flexible high speed logic solutions by directly mapping algorithms into the hardware fabric in an FPGA chip. Kale and Shriramwar [1] for example used FPGAs to create highly flexible control algorithms for wheeled robots. This approach of FPGA-based development can offer higher performance in the final system. However, the improved performance is achieved at the penalty of high development costs. The high cost is due to the manual development of the register transfer level (RTL) code for an algorithm and the huge debugging overhead introduced by the long synthesizing and routing time (normally 30 minutes) .

To improve the efficiency of FPGA-based system development, a set of general purpose soft RISC processors have been developed and delivered by FPGA vendors. For example, Xilinx's MicroBlaze™ core [2] and Altera's Nios II cores. These soft cores and the corresponding compilers provide software programmability that system developers normally see in microcontrollers. Therefore the development process is accelerated by reusing existing software, avoiding RTL programming and avoiding frequent synthesizing hardware during debug time.

Eugenio and Estradase [3] made use of the higher functionality and flexibility introduced by a soft computing core implemented on an FPGA and combined this with a Linux OS making for a highly configurable and easily programmable design. Following in the same line several other research projects were conducted using FPGAs in different robots. [4][5].

This improved development efficiency comes at the cost of performance particularly in terms of parallelism. To improve the system performance with the reduced development cost, a configurable multi-core architecture that can address the application-level parallelism in the hardware is often required.

Shimai and Tani et al [6] describe a multi core mixed integer quadratic programming solver for mobile robot control. Such multi core parallel computing platforms are essential as the number of parallel tasks increase and a problem may be solved by breaking it up into multiple tasks. In this case the processing is achieved by multiple dedicated computation cores. Sun et al [7] describe a dual core robot controller based on Altera's Nios II core and Cyclone II FPGA. By specifying a switching fabric to a shared memory, the two processors may execute in parallel on shared data. However in [7], the size of the cores and the prefixed interconnect architecture (the shared memory) limits the parallelism that can be mapped into the hardware. The size of a core limits the number of cores that can be

978-0-7695-4694-0/12 $26.00 © 2012 IEEE
DOI 10.1109/ICIS.2012.12

implemented on an FPGA chip, the interconnect architecture decides the communication overhead among the cores. That is, the smaller the core the smaller the parallelism granularity. As a result more parallel tasks in the application can be directly mapped to parallel cores. In addition the closer the interconnect architecture is to the communication path in an FPGA application, the lower the communication overhead on the final system and an application-tailored interconnect will offer better performance on FPGA-based systems.

To match this smaller parallelism granularity a high-level programming environment that contains an efficient hardware library, a programming language and an intelligent compiler is required to achieve the development and runtime efficiency. The tools provided in the programming environment should map the application-level parallelism into processor cores and the communication path into suitable interconnect architectures implemented in the hardware library.

In the following section 2 describes the use cases covered by our work. Section 3 describes the FPGA hardware while section 4 explains the co-design architecture. Section 5 and 6 discuss our results and future work.

2. Reconfigurable requirements of the use cases

The use cases that we have studied include sensors for e-Health and highly mobile robots. The use cases make different demands on the reconfigurable FPGA technology and therefore are ideal to determine if the approach is generic enough to solve problems from diverse domains.

2.1 E-Health Use case

The work with e-health sensors has been stimulated by an EU funded project called OPTIMI, in which multiple sensors were developed to monitor heart signals, brain signals and physical activity. These sensors can operate in two modes. In the first mode they detect the signals and process them locally. However in this mode the local processors have just enough computational power to do simple algorithmic tasks. In a second mode the sensors stream data to a secondary processor where more complex processing is performed.

An example use case is a group relaxation session for 5 patients each of whom wears an OPTIMI Electro-encephalograph (EEG) sensor. Each user's brain signals are processed in order to determine their level of relaxation and to compare each one to each other in the group. To achieve this, the raw brain wave signals from the 5 users are sent wirelessly in real time to a central processor unit. The unit performs parallel FFT analysis on each stream, generates the relaxation results, compares them and sends the results for display.

Figure 1 Parallel Streams of EEG data

Achieving this with conventional PC based systems is not a simple matter especially when each stream must be processed with zero inter stream latency. Each stream FFT result (Alpha, beta, Delta and Theta) as seen in Figure 1, must be compared simultaneously and the results sent for visualization in real time.

2.2 Robotics Use case

The bipedal robot shown in Figure 2 has 7 moving limbs; the feet, the lower legs, the thighs and the hip. Each limb has 4 or more actuators controlled by an I/O processor as well as accelerometers, limb angle sensors and pressure sensors.

In order to control this robot, the central processor must process a total of 60 sensor signals and generate 30 output control signals. Since the robot has 14 degrees of freedom and actuation forces are highly nonlinear, classical control methods are used only at a low level such as limb angle control.

The overall control is performed using a machine learning technique in which a very large number of trained rules are used to stabilize the robot as well as maneuver it. In contrast to the EEG sensor use case, in this case the processor uses streaming parallelism only at the i/o processor level and then combines this data into a hierarchical solution made up of 3 layers of control. These layers include clustering based feature extraction, state classification and goal direction.

The computation resources required for each layer is different. Machine Learning requires the most resources due to the complex mathematical data processing and data processing may occur over several minutes before results are obtained. Goal generation combines external user command data with local functionality while state classification uses the rules generated by the machine learning to identify states and provide reactionary set points. Although the data processing is less complex in these two layers the cycle times are very short and in the order of 3 to 5 milliseconds. Finally the i/o streams may be handled by dedicated processing elements that route data to a shared memory.

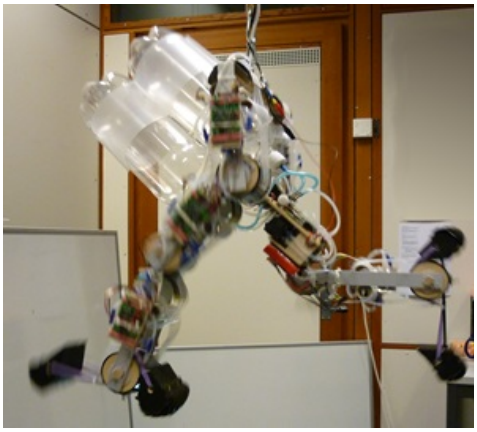

Figure 2 Running Robot

3. Hardware

The FPGA implementation may be described first at the level of the prototype board incorporating the FPGA, and I/O devices. Then the internal aspects of the FPGA may be described explaining the multi core software hardware co-design.

3.1 Prototype board

The prototype board made to interface the robot and the e-health sensors is based on the Virtex 5 FPGA from Xilinx [2]. The board has been designed to be as simple as possible and the intention is to position it as a wireless multichannel multi-core stream processing engine.

The RF communications is achieved using 7 channels of Nordic Semiconductor nRF24L01 devices running at 2.4GHz and achieving a theoretical maximum of 2Megabits per second. The RF Front ends are driven over SPI directly from the FPGA. The board also has an RS232 port, a 16Mega Byte SPI FLASH memory, a memory retention battery and a 75MHz clock.

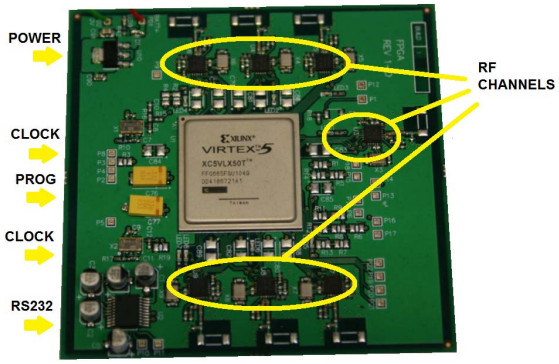

Figure 3 The FPGA prototype

3.2 FPGA

The many-core processor is built from the hardware library implemented on an FPGA that includes the following components. For computation a tiny register machine (TRM) with a 2-stage pipelined implementation running at 116MHz. This takes up 2% of the LUTs of the Virtex-5XC5VLX50T FPGA. For storage a DDR2 controller interface for a Xilinx ML505 board has been implemented. It runs at 116MHz with DDR2 clock 223MHz, and takes 2% LUTs and 5% BRAMs.

Communication components consist of buffered channels implemented as FIFOs with different width and depth configurations. These channels are used to transfer data from one processor core to another.

The floating point unit has been implemented using 2% LUTs of the Virtex-5LX50T. It takes 8 cycles to do a floating point addition and 4 cycles to do a floating point multiplication. Floating point division is emulated in software.

For the I/O controllers there is a compact flash (CF) controller, an LCD controller and a UART controller all running at 116MHz internally. A VGA controller and a DVI controller have also been implemented. They run at pixel clock rate.

4. System and Software Architecture

The main aim of this paper is to show the ease with which the two application use cases may be achieved.

The application software is written in the high level language OBERON [9], for which the authors have developed a full development environment that is fully integrated into the Xilinx ISE Design Suite [10].

Having made sure that the target FPGA is well specified to the development environment, a new developer simply writes an OBERON code to immediately instantiate as many parallel cores as required, which we have called Cells, and then links the Cells with an interconnect. OBERON is then used to write the code for each Cell just as normally one would write any code. So in some cases the Cell code may be very particular for that Cell or it could be a general purpose code written such that many parallel or sequential executing instantiations of the Cell may execute simultaneously at run time.

4.1 EEG data stream processing

The EEG processing problem is split into several parallel streams and the results from each stream are gathered for display. In figure 4 below one can see diagrammatically, three parallel streams 1, 2 and 3.

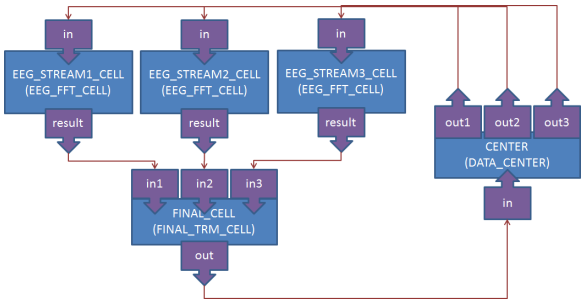

Figure 4 The EEG Parallel Stream Case

The multi Cell (multi-core) design is as follows. A data centre provides SPI engines that allow data to be read and written to the RF front ends.

Data from the 3 EEG sensors is read from the data centre and routed to 3 active cells. Each active cell implements a 512 point floating point FFT on the data.

The results are sent to a final active Cell to group the results. The overall result is then sent over RF to a waiting display via the SPI engines in the data centre. To define the active Cell or core, we would use the following code fragment.

```
(* Define a new Cell and associated code for implementing the EEG FFT
stream processing *)
TYPE EEG_FFT_Cell=
CELL (in: STREAM {IN} OF REAL; result:  STREAM {OUT} OF REAL )
{ DataMemorySize=4096, FloatingPoint}

(* Here we would put the math for the FFT procedure Do_FFT... *)

BEGIN
  RECEIVE(in, realIn, imagIn);   (* Read in the stream from port "in" *)
  Do_FFT(NumSamples, realIn, imagIn, realOut, imagOut);
  SEND(result, realOut, imagOut); (* Output results to the port "result" *)
END EEG_FFT_Cell;
```

The above fragment defines a generic Cell that implements an EEG FFT on its array input and generates an array of output. All the code relating to the math of the FFT is not shown for brevity but would be placed where there is the comment.

In order to gather the data from each EEG FFT Cell, a fourth Cell is required to perform the final data gathering.

```
(* Define a new cell for implementing the results gathering and
forwarding *)
FINAL_CELL=CELL(in1,in2,in3,in4,in5: STREAM {IN} OF REAL; out:
STREAM {OUT} OF REAL)
BEGIN
(* Combine final result here and send*)
END FINAL_CELL
```

In the above code fragment this gathering cell is clearly defined with multiple input arguments and a single output argument.

Data must be interfaced to the outside world via SPI engines that talk to the RF frontend. This layer is implemented by the data center which reads input streams and routes them to the EEG FFT cells.

```
DATA_CENTER=CELL(in: STREAM {IN} OF REAL; out1, out2, out3,
out4,out5:  STREAM {OUT} OF REAL)
BEGIN
(* feed in data here and read the result *)

END DATA_CENTER
```

In order to instantiate the 3 parallel EEG FFT cells, the gathering cell and the data centre first they are declared as objects and then instantiated along with interconnection definitions. The following is used.

```
VAR EEG_STREAM1_CELL: EEG_FFT_Cell;
VAR EEG_STREAM2_CELL: EEG_FFT_Cell;
VAR EEG_STREAM3_CELL: EEG_FFT_Cell;
VAR FINAL_CELL: FINAL_TRM_CELL;
VAR  CENTER: DATA_CENTER;
BEGIN
(* specify connection & layout *)
          NEW(EEG_STREAM1_CELL);
          NEW(EEG_STREAM2_CELL);
          NEW(EEG_STREAM3_CELL);

CONNECT(CENTER.out1,EEG_STREAM1_CELL.in );
CONNECT(CENTER.out2,EEG_STREAM2_CELL.in );
CONNECT(CENTER.out3,EEG_STREAM3_CELL.in );

CONNECT( EEG_STREAM1_CELL . result , FINAL_CELL .in1 );
CONNECT( EEG_STREAM2_CELL . result , FINAL_CELL .in2 );
CONNECT( EEG_STREAM3_CELL . result , FINAL_CELL .in3 );

CONNECT(FINAL_CELL .out, CENTER.in);

END TEST_FFT.
```

The above code implements the 5 core application as 5 active Cells. Shared memory is replaced by a streaming data approach in which only necessary data is passed between the parallel applications.

4.2 Robot Hierarchical processing

As described earlier the robot is controlled via a hierarchic layered approach. Of course this still requires a high level of parallelism and a multi core approach.

To implement the robot controller we require a cell for feature extraction and state classification task, a cell for the goal direction and a cell for machine learning of robot behavior. In addition data must be gathered from the remote SPI to RF components.

A layer is also required to buffer data between the I/O processors and format it correctly for the main computation layers.

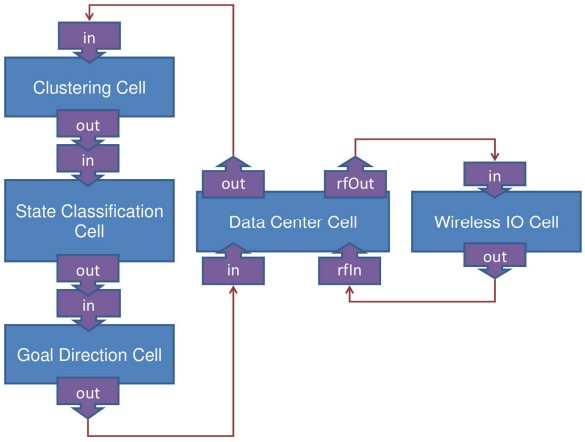

Figure 5, Layered cell structure for the running robot

The above Figure 5 depicts these layers. The code fragment below shows the cell definitions.

```
CELLNET RunningRobot;
(* Define the Machine learning TRM CPU and associated code *)

TYPE

WirelessIOCell = CELL (in: PORT IN; out: PORT OUT)
(* Here we would put the SPI procedures *)
BEGIN
   RECEIVE(in, pattern);   (* Read in the stream from port "in" *)
   (* Drive the wireless-IO-chips over SPI *)
   SEND(out, measurements); (* Output the data from the sensors *)
END ClassificationActuationCell;

DataCenterCell = CELL(              rfIn: PORT IN; rfOut: PORT OUT;
                                    in: PORT IN; out: PORT OUT)
(* shared memory access point *)
BEGIN
   (* feed in data here and read the result *)
END DATA_CENTER

ClusteringCell = CELL (in: PORT IN; out: PORT OUT)
BEGIN
   RECEIVE(in, realIn);   (* Read in the stream from port "in" *)
   DoClustering(realIn, quantization);  (* Process the data *)
   SEND(out, quantization); (* Output the results to the port "out" *)
END MachineLearningCell;

StateClassificationCell = CELL (in: PORT IN; out: PORT OUT)
BEGIN
   RECEIVE(in, quantization);(* Read in the stream from port "in" *)
   DoClassification(quantization, classification);(* Process the data *)
   SEND(out, classification); (* Output the results to the port "out" *)
END MachineLearningCell;

GoalDirectionCell = CELL (in: PORT IN; out: PORT OUT)
(* Here we would put the Machine Learning procedures *)
BEGIN
   RECEIVE(in, state);   (* Read in the stream from port "in" *)
   DoGoalDirection(state, pattern);  (* Process the data *)
   SEND(out, pattern); (* Output the resulting pattern to the port "out" *)
END GoalDirectionCell;
```

The code fragment below shows the instantiation of the cells.

```
VAR
   wirelessIO: WirelessIOCell;
   dataCenter: DataCenterCell;
   clustering: ClusteringCell;
   classification: StateClassificationCell;
   goalDirection: GoalDirectionCell
```

Finally the code below shows the Cell interconnect.

```
BEGIN
   NEW(wirelessIO);
   NEW(dataCenter);
   NEW(clustering);
   NEW(classification);
   NEW(goalDirection);

   CONNECT(wirelessIO.out, dataCenter.rfIn);
   CONNECT(dataCenter.rfOut, wirelessIO.in);

   CONNECT(dataCenter.out, clustering.in);
   CONNECT(clustering.out, classification.in);
   CONNECT(classification.out, goalDirection.in);
   CONNECT(goalDirection.out, dataCenter.in);
END RunningRobot.
```

5. Results

The FPGA prototype has been built and tested and the applications, from which the code fragments are taken were implemented.

In the case of the robot controller, the main results have so far been obtained in communicating between the FPGA prototype and the robot I/O processors. The communications over the RF front ends work reliably with all cells operating in parallel. Although the machine learning, goal direction and classification cells are still under construction, the basic architecture has been shown to operate well using dummy cells.

The SPI to nRF24L01 front end has been measured to transfer 120KB/s of payload per channel. Internally the interconnect allows data to be passed up and down the layers such that n bytes may be circulated at 100MB/s with a latency of 50ns down to 10ns.

In the case of the EEG data processing, again the multi cell approach was implemented and tested. It was found that at 100Mhz synthesized internal clock the 512 point floating point FFT may be completed within 9.52ms.

When used at a sample rate of 512 Hz and using a 64 point FFT, the EEG data from 7 patients can be processed fully in real time, each FFT taking 1.16ms to calculate.

7. Conclusions

The need for simple, efficient and highly parallel reconfigurable computing devices is being answered by the software and hardware co-design on FPGA devices.

Central to the approach is an integrated development environment in which in this case the high level language OBERON is compiled down to Bit Stream data, and where the OBERON is running on language defined tiny RISC processors.

A multi cell approach and an interconnect allows very complex parallel streamed applications to be developed as well as hierarchical layered parallel applications.

The co-design has been applied to two very different domains using a common wireless and FPGA merged approach reliant of off board I/O processors or sensors and actuators.

Acknowledgement

This work was performed with funding from MA Systems and Control Limited and all hardware has been designed at MA Systems in the U.K.

References

[1] (IJCSIS) International Journal of Computer Science and Information Security,Vol. 3, No. 1, 2009; FPGA-based Controller for a Mobile Robot, Ms. Shilpa Kale Dept. of Electronics & Telecommunication Engg. Nagpur

[2] http://www.xilinx.com/tools/microblaze.htm.

[3] J. Moctezuma Eugenio and M. Arias Estrada; Hardware Software FPGA Architecture for Robotics Applications, Reconfigurable Computing: Architectures, Tools and Applications; Lecture Notes in Computer Science, 2009, Volume 5453/2009, 27-38

[4] J. Alves and N. Cruz; An FPGA-Based Embedded System for a Sailing Robot, 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009.

[5] C. Castro , C. Llanos, W. de Britto Vidal Filho and L. dos Santos Coelho; Fuzzy Control for Cyclist Robot Stability using FPGAs International Conference on Reconfigurable Computing and FPGAs, 2009.

[6] Y. Shimai, J. Tani, H. Noguchi, H. Kawaguchi, M. Yoshimoto; FPGA implementation of mixed integer quadratic programming solver for mobile robot control; International Conference on Field-Programmable Technology, 2009. FPT 2009.

[7] Ligong Sun, Xiangwen Sun, Fei Xiang and Sujuan Li; Design of Dual-Core Architecture Industrial Robot Controller Based on FPGA; Advanced Materials Research Vols. 291-294 (2011) pp 3287-3291

[8] N. Wirth and M. Reiser; Programming in Oberon - Steps Beyond Pascal and Modula; Addison-Wesley, 1992, ISBN 0-201-56543-9

[9] Ling Liu, Oleksii Morozov, Yuxing Han, Jürg Gutknecht, and Patrick R. Hunziker. Automatic soc design on many-core processors: a software hardware co-design approach for fpgas. In FPGA, pages 37-40, 2011.

[10] http://www.xilinx.com/ise_eval/