Adaptive Systems

Astha Lagoo[1] Rohan Kumar[2], Shivam Sharma[3], Tanmay Govil[4], A.A. Deshmukh[5], Ms Priti Ranadive[6], Mr.Chaitanya Rajguru[7]

University of Maryland, College Park, MD, USA
[1, 2,3,4]students Smt. Kashibai Navale Collge of Engg. Department of Computer Engg, Pune University
[5]Faculty Smt. Kashibai Navale Collge of Engg. Department of Computer Engg, Pune University
[6, 7]Ph.d. Fellow,Department of Electronics Systems , Aalborg University, Denmark 9220
[4]KPIT Cummins India

Abstract
Proposed project will implement an algorithm both on hardware as well as software. It will then determine the performance of both the hardware platform as well as the software one. The results will then be compared to determine which implementation was more efficient. It will then divide the algorithm into parts and execute them separately on h/w and s/w and will try finding the optimum ratio. The proposed project will also determine which hardware device is to be used for the hardware implementation.

1. Introduction

As the demands are growing computer systems are becoming increasingly complex. For example, a good mobile phone today is expected to take high-resolution pictures, video and audio, play back music files in various formats, manage user files such as presentations, run internet browsing software, connect using multiple wireless protocols such as GSM, GPRS, CDMA, Wi-Fi, WiMax and Bluetooth, and of course make phone calls. In the coming time many more applications will be developed with advanced features to run in this size- and power-constrained computing system. Multiple simultaneous applications are already the norm. This can be achieved with the use of more hardware and software but that will increase the cost, size and power. Alternate approaches will thus be important in meeting the growing user needs.

The presented scheme includes:

- A flexible system that can implement different hardware and software configurations, and
- A supervisor tool that can take input from the user regarding their preferences, and that can also monitor. the system usage. This tool will then adapt the system configuration dynamically in response to dynamic compute demands.

Other collateral developed will include new tools, methods and capabilities needed to enable system adaptation, including modified versions of application software & hardware.

This project will represent a leading-edge exploration of the latest trends in computing. The innovative content of this project is a dynamically self-modifying system that adapts itself to the changing usage demands, which will enable lower-cost platforms that satisfy a wide range of needs.

It is hoped that this project will advance the knowledge and capabilities for hardware and software development on computing systems. We expect that this created knowledge will lead to the development of novel computers. It will be disseminated in the form of papers.

The proposed project will examine the use of FPGA systems in a continuously adaptive manner. By adaptation, we mean that the system hardware & software adapts itself to the changing workload demands. An example of hardware adaptation is enabling or disabling certain portions of the hardware, or modifying the hardware logic using FPGA or other reprogrammable logic. This project will develop new techniques to extend the benefits of reconfigurable systems in real-life systems with varying usage demands.

The configurable logic place and route step is the most computationally intensive part of such hardware/software partitioning, normally running for many minutes or hours on powerful desktop processors. In contrast, dynamic partitioning requires place and route to execute in just seconds and on a lean embedded processor. We have therefore
designed a configurable logic architecture specifically for dynamic hardware/software partitioning. By specifically focusing on the goal of software kernel speedup when designing the FPGA architecture, rather than on the more general goal of ASIC prototyping, we can perform place and route for our architecture 50 times faster, using 10,000 times less data memory, and 1,000 times less code memory, than popular commercial tools mapping to commercial configurable logic. Yet, we show that we obtain speedups (2x on average, and as much as

Manuscript received July 5, 2011
Manuscript revised July 20, 2011

4x) and energy savings (33% on average, and up to 74%) when partitioning even just one loop, which are comparable to commercial tools and fabrics. Thus, our configurable logic architecture represents a good candidate for platforms that will support dynamic hardware/software partitioning, and enables ultra-fast desktop tools for hardware/software partitioning, and even for fast configurable logic design in general.

2. LITERATURE SURVEY

A. Background Details
1) Adaptive Systems:
A part of the application can be run either on hardware or software. Adaptive System are system's that are sensitive to the present conditions prevailing and can decide according to the conditions, which part to run on hardware and which to run on software. Hardware part of the application is implemented using FPGA (FPGAs).

2) H/W S/W Partitioning:
Embedded systems are designed to implement specific applications as efficiently and effectively as possible. Once a system is designed and implemented modification of the system can only be achieved in the reconfiguration of the system's software as the hardware is fixed. Hardware flexibility is now obtainable through the use of FPGAs

3) FPGA
An FPGA (FPGA) is a type of programmable chip that can be made to behave in just about any way you wish. The FPGA must be configured before it can be used; The configuration process is very similar to programming a computer, except that instead of making a software program behave a certain way on a computer, you're making a chip behave a certain way.

4) CRC
A CRC is an error-detecting code. Its computation resembles a long division operation in which the quotient is discarded and the remainder becomes the result. The definition of a particular CRC specifies the divisor to be used. The simplest error-detection system, the parity bit, is in fact a trivial CRC: it uses the two-bit-long divisor 11. It is discussed in detail in the next section.

B. Conclusion
Most of the paper's on adaptive systems focus on dynamic reconfiguration of h/w and s/w in order to achieve an optimise performance. In the paper the case for reconfigurable hardware in wearable computing advantages of reconfigurable architecture in wearable computing are being emphasized. But in our paper we mainly focus on executing an algorithm on h/w and then on s/w and then comparing the results obtained. After which we try to determine an optimum combination running algorithm both on s/w as well as h/w. [7]

3 SYSTEM STUDY

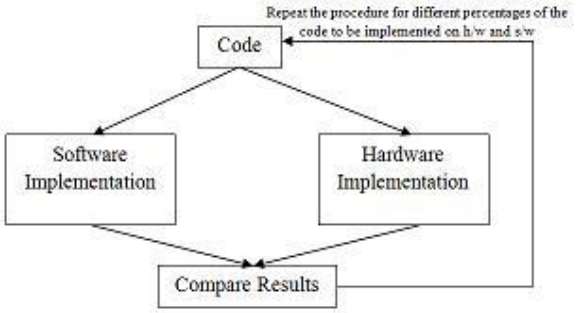


Fig. 1 Block diagram of the system

A field-programmable gate array (FPGA) is a configurable integrated circuit. It can be configured using a hardware descriptive language (HDL) either by the customer or the developer after manufacturing. FPGAs offer much lower non- recurring engineering costs compared to application-specific integrated circuit (ASIC) and can be used to implement any logical function that an ASIC could perform.
FPGAs contain programmable logic components called logic blocks, which can be connected using a hierarchy of reconfigurable interconnects. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. Our underlying model of architecture is a single processor host connected to a reconfigurable FPGA board system through a bus. The program initially is in software and contains a set of functions which can either be mapped onto the FPGAs or executed in the processor itself. To configure the FPGA we use the standard Xilinx software and load the program onto the FPGA. We then run the CRC code and note down the power and performance.
For the software execution of the code we use a P4 processor and the Turbo compiler. We are using Windows XP operating system and boot the system in Safe Mode with command prompt only in order to curb the effect of background processes on the profiling results. The code is
then run and the power and performance is noted. This process of noting down the power and performance is then repeated for different percentages (eg. 30% in h/w and 70% in s/w) of the code run in h/w and s/w. The results are then compared to find out the most optimised ratio in which code should be divided in between the h/w

and s/w implementation. This result gives us the minimum power and maximum performance.

4. CRC

Cyclic redundancy check (CRC) is used to detect accidental changes in data. CRC is commonly used while transferring data through networks and also while transferring data from secondary memory (hard disks) to primary memory. A CRC code can only be used to detect errors and not correct them. A CRC-enabled device calculates a short, fixed-length binary sequence, known as the CRC code or just CRC, for each block of data and sends or stores them both together. When a block is read or received the device repeats the calculation; if the new CRC does not match the one calculated earlier then the block contains a data error and the device may request the sender to send the block again.

C. Hardware Implementation
5) Hardware required for implementation:
A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, and the low non-recurring engineering costs relative to an ASIC design (not withstanding the generally higher unit cost), offer advantages for many applications.

6) Ways to configure FPGA:
With a cable attached to a computer Hooking up a cable to a PC's serial or parallel port and having it configure the FPGA through specially-made software is probably one of the simplest solutions. You design your FPGA on the computer screen and generate the code there, then when you're done you just attach the cable and the computer configures the FPGA for you
Assuming that you are doing things the "normal" way and connecting your FPGA to a computer, the following is a list of the basic steps needed to program your FPGA:

1. Write the HDL (Hardware Description Language) code HDLs are high-level languages, and they will actually be compiled into binary data before they are loaded into the FPGA. HDLs are usually either Verilog or VHDL. Verilog files have a .v extension, while VHDL files have a .vhd or .vhdl extension. [6]

2. Assign pin names. When you program in an HDL, you use names to refer to the pins of the FPGA. The compiler just needs to know what name corresponds to what pin. For example, suppose you want pin 10 on the FPGA to be called "input3" in your HDL code. The compiler needs to know that "input3" refers to that pin. [6]

3. Compile the FPGA's binary file using the FPGA manufacturer's proprietary software. Xilinx' software for FPGA development is called ISE. [6]

4. Transfer the bit stream into the FPGA. How you do this will depend on what method you're using to connect to the FPGA, but assuming that you've got it connected to your computer, you'll need some interface utility to upload the binary file to the FPGA. [6]
We port the code in VHDL to FPGA through a wired connection

7) Implementation:
Serial CRC Checksum generation circuit for FPGA implementation

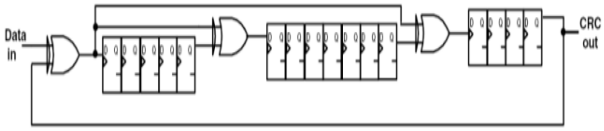


Fig. 1 Circuit for CRC checksum generation

Here we take a shift register, having the same size as that of the key (actually 1 less, considering the top bit of the key as implicit 1) that is being used for CRC calculation. The shift register is initially initialised with all 0 bits. The input (no of zeroes are added to the input depending on the size of key) is moved bitwise into the shift register. After moving a single bit of input into the register it is checked whether the top bit of the shift register is 1 or 0. If 0 then the shift register is left as it is and the next input bit is moved in. If 1 then the contents of the shift registers is xor'd with the key. The above procedure is repeated till the input is finished. The final content's of the shift register is the CRC code. [2]

D. Software Implementation of CRC

1) Modulo 2 division:
The number to be divided is the message appended with zeros at the end. The number of zero bits added to the message is the same as the width of the checksum. Here the check sum is being called as c. In this case four bits are added (since the size of the key is four). The divisor is a c+1-bit number known as the generator polynomial (key). [9]
The modulo-2 division process is defined as follows [9]:

- Shift the uppermost c+1 bits of the remainder of the message.

- Starting with the top most bit in the original message and for each bit position that follows, look at the c+1

bit remainder:
o If the most significant bit of the remainder is a one, the divisor is said to divide into it. If it happens the following will be done:
  - Set the appropriate bit in the quotient to a one, and
  - XOR the remainder with the divisor and store the result back into the remainder

o Otherwise when the first bit is not one:
  - Set the appropriate bit in the quotient to a zero, and

o XOR the remainder with zero
o Left-shift the remainder, shifting in the next bit of the message. The bit that's shifted out
will always be a zero hence no information is lost.
The final value of the remainder is the CRC of the given message.

2) Pseudo Code:
The pseudo code for the software implementation is written below:

- Store the value of the generator polynomial (key) in an integer variable. The integer variable can be short or long depending upon the size of key.
- Similarly store the input after augmenting zero's in an integer variable and also declare one to represent the shift register while also initialising it to zero.
- Send the input byte by byte to the put byte function.
- The put byte function will convert the input to bits and send it bit by bit to the put bit function.
- The put bit stores the bits that it is receiving one by one into the shift register while moving the top bit out (shifting process). Then it checks whether the top bit is 0 or 1. If it is 0 it leaves the shift register as it is. But if it is 1 it xor's the contents of the shift register with the key.
- The above procedure is repeated till all the bits of the input have been sent to the put bit function. The final content's of the shift register is our CRC checksum.

5. PROFILING RESULTS

The profiling results of hardware and software implementation of CRC algorithm are written below.

A. Software Implementation
1) Performance: To measure the performance of the algorithm time function of the C language was used. The program was made to loop 10000 times. The time taken for execution was calculated and then divided by 10000. On doing so it was observed that the time taken for the execution of program was 1.483516E-3. The clock ticks were 27. [8]

2) Power: To measure power time taken by the program and the statistics available on the CPU datasheet of the P4 processor obtained from the Intel website was used. a 2.2 GHz P4 with a 400 MHz FSB has a typical Vcc of 1.3725 Volts and Icc of 47.9 Amps which is (1.3725*47.9=) 65.74 watts. Since we know the loop of 10,000 algorithm cycles took
1.483516s, we can assume a single loop will take
1.483516/10000 = 0.0001483516s. The amount of energy consumed by the algorithm would then be 65.74 watts *
0.0001483516s = 0.0098 watt seconds (or joules).[1]

B. Hardware Implementation
1) Performance:
In the CRC program every one bit input requires one clock pulse. At the rising edge of the clock pulse the bit enters the shift register. By this we will get to know the number of clock pulses required to find the CRC checksum for the particular input. The first clock pulse is used for resetting and enabling

the CRC procedure, at the next clock pulse first bit of input will be inserted and then the next bit and so on.
Now we know that the number of clock pulses required will be
=N+1
Where N is number of input bits

Referring to the Xilinx Spartan 3 family Datasheet it was known that time taken by one clock pulse is 9.104ns

Now time taken by the FPGA to execute and find the CRC Checksum for an N bit input is
the expected current per voltage source required. This view includes both off-chip and on-chip dissipated power. [5]
• Power by User Logic Resources - For each type of user logic in the design, XPE reports the expected power. This allows you to experiment with architecture, resources, and implementation trade-off choices in order to remain within the allotted power budget. [5]
• Thermal Power-XPE lets you enter device environment settings and reports thermal properties of the device for your
application, such as the expected junction temperature. With this information you can evaluate the need for passive or active cooling for your design. [5]

((N+1) * 9.104) ns.

If input is of 16 bits
=((16+1)*9.104)ns
=154.768 ns

1) Toggle Rates:

• For synchronous paths, toggle rate reflects how often an output changes relative to a given clock input and can be modelled as a percentage between 0–100%. The max data toggle rate of 100% means that if the clock frequency is 100
MHz, a data toggle rate of 100% equates to a data frequency of 50 MHz. This convention stems from an assumption that a

2) Power:

Xilinx Power Estimator is used for calculating the power used by FPGA for the particular CRC program.
The Xilinx Power Estimator (XPE) spreadsheet is a power estimation tool typically used in the pre-design and pre- implementation phases of a project. XPE assists with architecture evaluation and device selection and helps in selecting the appropriate power supply and thermal management components which may be required for your application.XPE considers your design's resource usage, toggle rates, I/O loading, and many other factors which it combines with the device models to calculate the estimated power distribution. The formulas used for calculations in the program are based on intended behaviour of various digital circuits. The device models are extracted from measurements, simulation, and/or extrapolation.[5]

XPE presents multiple views of the power distribution.

• Static vs. Dynamic Power - Static or Quiescent power represents the power drawn when the device is powered and programmed and there is no switching activity. This includes transistor leakage, power consumed internally, and power dissipated in external termination resistors. Dynamic power is the additional power consumed or sourced by the FPGA when the user logic is active. This description is typical for an FPGA technology based on CMOS SRAM transistors. By choosing the Maximum or Typical values for Process, worst- case or nominal power can be estimated.[5]
Both the reported static and dynamic power estimates are modelled to account for temperature and voltage sensitivity. Ambient temperature and regulated voltage on the system can be keyed into the appropriate cells provided for the purpose. [5]

• Power by Voltage Supplies - This view is useful to select the appropriate voltage supply sources since XPE breaks out net with a 100% toggle rate toggles every active edge of a clock whose frequency is specified. This means that the LSB of a 16-bit counter toggles every clock cycle and the MSB toggles every 32,768th clock cycle. [5]

• For non-periodic or event-driven state machine designs, toggle rates cannot be easily predicted. An effective method of
estimating average toggle rates for a given design is to segregate the different sections of the design based on their functionality and estimate the toggle rates for each of the sub-
blocks. An average toggle rate can then be arrived at by calculating the average for the entire design. Most logic-
intensive designs work at around 12.5% average toggle rate,
which is the default toggle rate setting in XPE. For a worst- case estimate, a toggle rate of 20% can be used. Average toggle rates greater than 20% are not very common. Arithmetic-intensive modules of a design seem to take toggle rates of up to 50%, which is representative of the absolute [5]

Power Supply Resources Powered [5]
VCCINT
- All CLB resources
- Most configuration SRAM cells
- All routing resources
- Entire clock tree, including all clock buffers
- Block RAM
- DSP blocks (DSP48, DSP48A, DSP48E)(1)
- All input buffers
- Logic elements in the IOB (ILOGIC/OLOGIC)(1)
- ISERDES/OSERDES(1)
- PowerPC™ processor
- Tri-Mode Ethernet MAC
- DCM (minor)

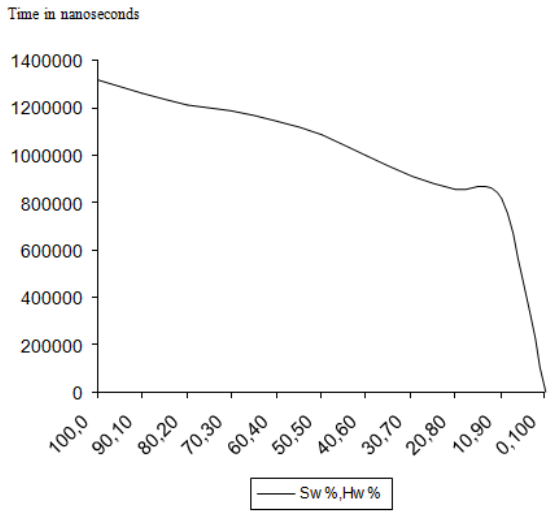
VCCAUX
- IODELAY/IDELAYCTRL
- Differential Input buffers
- VREF-based, single-ended I/O standards, e.g.,

HSTL18_I
- Clock Managers (DCM, PLL, PMCD)
- Some Configuration memory

VCCO
- All output buffers
- Some input buffers
- Digitally controlled impedance

on y-axis. Then we plot lines for software and hardware implementation on the graph and also a line for the combined value. At the point we get the lowest value on this line we note the percentages.
This is the optimum value of hardware software

partitioning ratio at which the CRC code takes the minimum time to run.



Voltage Source Information				
Source	Voltage	Power (W)	I_{CC} (A)	I_{CCQ} (A)
V_{CCINT}	1.2	0.021	0.002	0.015
V_{CCAUX}	2.5	0.038	0.000	0.015
V_{CCO} 3.3	3.3	0.000	0.000	0.000
V_{CCO} 2.5	2.5	0.008	0.002	0.002
V_{CCO} 1.8	1.8	0.000	0.000	0.000
V_{CCO} 1.5	1.5	0.000	0.000	0.000
V_{CCO} 1.2	1.2	0.000	0.000	0.000

Using Xilinx Power Estimator we get: Hence, power consumed is **67 mW**.

Power Summary	
Optimization	None
Data	Production
Quiescent(W)	0.060
Dynamic (W)	0.007
Total (W)	0.067

6. Compare Results

After getting the number of clock cycles taken by the software as well as the hardware implementation of the CRC code we compare it with the help of a graph. To compare we calculate the values for running different percentages of the code in hardware and software. We start with 100% implementation in software, then 90% in software and 10% in hardware and so on till 100% in hardware.

After getting these values we plot the graph with hardware/software percentages on the x-axis and time taken

Sw%,Hw%	100,0	90,10	80,20	70,30	60,40
Time in ns	1318681	1263954	1213018	1187326	1145982

50,50	40,60	30,70	20,80	10,90
1090574	1002364	910846	853694	816323

0,100
1820

Note: This is the total time taken to find the checksum for the message, i.e. time take by software to find the checksum for the particular portion of the message + time taken by hardware to find the checksum.
Here, the message used is of 200bits.

7. Conclusion

After comparing the performance and power consumption of the execution of an algorithm on software and hardware we can say that high execution weight algorithms should preferably be executed on hardware. Though this approach
will not necessarily give an optimal solution but it will be close to an optimal one.
Algorithm which is to be executed on an adaptive system should be divided into tasks. Then the tasks can be categorised as high and low execution weights and the one's categorised as high should preferably be executed on hardware.

REFERENCES

[1] Eric Senn, Nathalie Julien, Johann Laurent, and Eric Martin, Power Consumption estimation of a c program for Data Intensive applications, Univ. of Massachusetts, Amherst, MA, and CMPSCI Tech.
[2] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/
[3] Aviral Shrivastava, Mohit Kumar, Sanjiv Kapoor, Shashi Kumar, M.Balakrishnan , Optimal Hardware/Software Partitioning for Concurrent Specification using Dynamic Programming, I.I.T. Delhi, New Delhi-110016, INDIA

[4] Mustafa Imran Ali ,Hardware/Software Partitioning and Scheduling Algorithms for Dynamically Reconfigurable Architectures
[5] Xilinx, Inc. http://xilinx.com
[6] N. Aviral Shrivastava, Mohit Kumar, ”Hardware Software Partitioning and Synthesis targeted towards FPGA based Implementation”, B.Tech Thesis, CSE Dept, IIT Delhi, May’99.
[7] Christian Plessl, Rolf Enzler, Herbert WalderJan Beutel, “the case for reconfigurable hardware in wearable computing”, 1 February 2003 / Accepted: 2 April 2003 / Published online: 11 September 2003.
[8] Intel, org. http://intel.com.
[9] Relisoft http://www.relisoft.com