Switching Activity Minimization by Efficient Instruction Set Architecture Design

Venkatraman Ramakrishna, Rakesh Kumar, Anupam Basu
Emails:{vrama@cs.ucla.edu, rakumar@cs.ucsd.edu, anupam@cse.iitkgp.ernet.in}

Abstract

Power consumption can be greatly minimized by reducing the bus signal transition activity (also called switching activity) in the control and data path circuit. Switching activity occurs due to the switching between two instructions (of the embedded software) on successive clock cycles. Our belief is that the binary encoding of instructions (machine code) plays a significant role in determining the amount of switching in a circuit. Thus, our aim is to realise a machine encoding of instructions of an ASIP such that for a given data path, it will minimize the average switching activity in the control path circuit of the ASIP and hence the total switching activity in the ASIP. Given the application-domain of the ASIP, we have used information theoretic techniques to arrive at an encoding of the op-code that minimizes redundancy and also the switching activity. We have compared our encoding of instruction op-codes with those obtained by other encoding techniques using a switching activity estimator designed by us.

1. Introduction

Power dissipation has become a critical *VLSI* design concern in recent years [3]. A substantial amount of research is being conducted at the *algorithmic* [3], *architectural* (such as *pipelining* [4] and *parallel-processing*) logic [5],[6], and circuit [7],[8] levels in order to develop power reduction techniques. Most of these efforts focus on reducing the on-chip dynamic power dissipation of CMOS circuits, which at a node is given by

$$P=(1/2)TCV_{dd}^2f \qquad (1)$$

where T is the transition activity at the node, C is the capacitance (the product TC represents the total switching capacitance), V is the supply voltage, and f is the frequency of operation. Because of appreciable C values, transition on the system busses result in considerable power dissipation. To address this problem various signal encoding schemes have been proposed in the literature [9], [10], [11], [12], [13], [14] to encode the data before transmitting on a bus so as to reduce the expected and peak number of transitions. Hence the signal encoding approaches in [9],[10],[11],[12],[13] and [14] achieve power reduction by reducing T in (1) while keeping C more or less unaltered.

In this paper, we propose an instruction-encoding scheme for ASIPs based on the popularity of instructions. We then show that our scheme results in lower switching activity in the controller circuit (for a given data path) as compared to some other standard encodings. Finally we state the limitations of this scheme and the state the direction that our future work might take.

2. Relating entropy and bus transition activity

In [1], upper and lower bounds on signal transition activity for any coding algorithm have been derived. **The bounds are asymptotically achievable if the process is *stationary* and *ergodic*.** We state their theorem (without proof) as follows: -

Theorem:

1) Let $\boldsymbol{H}$ be the entropy rate of a stationary and ergodic process {Xi}.

2) The symbols be coded in a uniquely decodable manner into bits represented by the binary random variables $B_1,B_2,....$, employing an expected number of $R(>\boldsymbol{H})$ bits/symbol.

3) The bits be transmitted in some arbitrary manner over a finite set of wires such that a receiver can uniquely decode the bits, and

4) T be the expected number of transitions in the bits on the wires per symbol, then

$$\boldsymbol{H^{-1}(H/R)\,R \le T \le (1-H^{-1}(H/R))\,R} \qquad \textbf{(5)}$$

Thus, the bounds on the switching activity are related to R, the number of bits used to encode a symbol. Hence, if we could decrease R by appropriate encoding, we can minimize the switching activity**.**

3. Application of Huffman's Algorithm in Reducing Switching Activity in an ASIP

The programs that are run on an Application Specific Instruction Processor (ASIP) are typically very similar in terms of the instructions constituting the program code. Hence, the probability of occurrence of an instruction (op-code) in a program can be found to a great deal of accuracy by getting frequency statistics for reasonably large number of benchmark programs.

Once we get the instruction probabilities, we can use these probability values for encoding the instruction.

The encoding we use for our instruction set architecture is *Huffman Coding with Zero Padding*.

Our algorithm for generating instruction op-codes takes a set S of tuples <*instruction, probability*> as input and outputs another set T of tuples <*instruction, machine code*>. The algorithm can be given as below:

```
Begin

//Let each instruction be kept in a inst_node, which
//contains the probability value of that instruction, and
//also two pointers (for children). Initially, the pointer
//values will be NULL. Keep the inst_nodes in a list
//sorted according to the probability value.

While (number of inst_nodes in the list > 1)
{
      Remove the two inst_nodes of least probability value from the list.
      Form a super inst_node whose probability value is the sum of the prob values of the two removed inst_nodes.
      Add this super inst_node to the list and sort the list.
      Make the super inst_node have the two removed inst_nodes as children.
}

// A tree of inst_nodes is formed, the root having prob.
//value 1.

Traverse the tree rooted at the one remaining inst_node in the list.
Assign a 0 to each left edge and a 1 to each right edge. (or vice-versa).

// Finally each instruction is represented at a leaf of
//the tree. The machine code corresponding to it is the
//bit string obtained on traversing the path from root
//to that leaf.

Get the size k of the largest instructions.

Pad all the instruction machine codes with 0s(or 1s) on the left such that size of all instructions is k.

End
```

As we can see, the machine codes of all instructions are forced to be of the same length by suitably padding (by 0s) the huffman codes at their MSB side.

4. Estimating Switching Activity in Sequential Circuits

Switching activity was estimated using the method given in [2].

Consider the set of functions below corresponding to the next state lines.

$$ns_1=f_1(i_1,i_2,...,i_M,ps_1,ps_2,..,ps_N)$$
$$...$$
$$ns_N=f_N(i_1,i_2,...,i_M,ps_1,ps_2,...,ps_N)$$

We can write:

$$prob(ns_1)=prob(f_1(i_1,i_2,...,i_M,ps_1,ps_2,..,ps_N))$$
$$...$$
$$prob(ns_N)=prob(f_N(i_1,i_2,...,i_M,ps_1,ps_2,...,ps_N))$$

where $prob(ns_i)$ corresponds to the probability that ns_i is 1 and $prob(f_i(i_1,i_2,...,i_M,ps_1,ps_2,..,ps_N))$ corresponds to the probability that $f_i(i_1,i_2,...,i_M,ps_1,ps_2,..,ps_N)$ is a 1, which is of course dependent on the $prob(ps_j)$ and the $prob(i_k)$.

We are interested in steady state probabilities of the present and next state lines implying that:

$$prob(ps_i)=prob(ns_i)=p_i \qquad 1 \le i \le N$$

The set of equations given the values of the $prob(i_k)$ becomes:

$$y_1=p_1-g_1(p_1,p_2,...,p_N)=0$$
$$...$$
$$y_N=p_N-g_N(p_1,p_2,...,p_N)=0$$

where the g_i's are non-linear functions of the p_i's. We will denote the above equations as $Y(P)=0$ or as $P=G(P)$. In general the boolean function f_i can be written as a list of *minterms* over the i_k and ps_j and the corresponding g_i function can be easily derived.

We can solve the equation set $Y(P)=0$ or find a fixed point of $P=G(P)$ to obtain the present state line probabilities.

Fig. 1. **k – unrolling of the next state logic**

To ensure better results, the next state logic is unrolled k (user-defined parameter) times. As illustrated in Fig. 1, we can construct a set of non-linear equations corresponding to this k-unrolled network, which will partially take into account the correlation between the state lines, when computing the state line probabilities.

To calculate the switching activity, we find the switching between the *k*th and the '*k+1*'th stage unrolled circuit and sum it over the *k* stages.

5. Design and Implementation

We designed an instruction set architecture based on the proposed encoding scheme. We also designed an estimator that takes a sequential logic circuit (and its input characteristics) as input and finds the corresponding switching. We fed implementations of different ISAs to the estimator to get switching activity statistics and compared our ISA with ISAs based on other encodings. (*For example*: Gray coding, Binary Coding).

5.1 Steps of Design

5.1.1 Design of Instruction Set Architecture

A minimal closed set of instructions was decided. The set is given below.

- sw Store word
- lw Load word
- add Add two registers
- blt Branch if less than
- bne Branch if not equal to
- ldi Load immediate
- sub Subtract two register values
- br Unconditional branch
- mov Move register to register
- bgt Branch if greater than
- beq Branch if equal to
- or Or two registers
- srl Shift right logical
- sra Shift right arithmetic
- sll Shift Left Logical
- ble Branch if less than or equal to
- and And two registers
- bge Branch if greater than or equal to
- not Not a register

DSP was chosen to form the domain of the ASIP.

Using various benchmark assembly (of Alpha-AXP) DSP programs (For example: Sort, Fibcall, Matrix Multiply, Compress, Laplace etc.) probability statistics of these instructions were calculated by a Statistical Analyzer.

5.1.2 Design and Specification of Control and Data Path

The *data path* was designed for execution of each instruction. The machine specifications are given below.

- 4-bit operands can be processed
- There are 2 registers
- Accumulator-based machine

We chose very simple specifications because the datapath was manually constructed.

The *controller* was designed using the instruction (machine) codes obtained above.

The control and data path obtained were now specified on a GUI called *Digisim*.

DigiSim is a Java-Swing based tool developed by us which allows specifications of circuits and ICs and saving them in standard formats (".ckt" and ".IC") designed for this purpose.

The net list corresponding to the specified circuit was extracted. (i.e. the interconnections among gates).

From the net list obtained above, the set of non-linear equations that describe the behavior of the circuit (as explained in [2]) were derived. Here we use the approximate estimation method suggested in [2]. In our case, we are feeding the instructions serially; hence the approximate method becomes an exact method, as the line inputs at every stage are now deterministic. So k-unrolling (for a k-cycle program) gives us exact switching statistics. (*Note: For measuring average switching activity of a circuit, this method is still approximate and to obtain exact values, unrolling must be done for* ∞ *number of times).*

We used Scilab [16] to solve these nonlinear equations.

6. Experimental Results

The switching activity results for four benchmark programs are presented in Fig 4, 5, 6 and 7. The switchings are presented for the control path circuit only. This is because switching in the data path circuit will remain constant irrespective of the machine encoding of instructions. Huffman coding (two sets, Encoding 1 and Encoding 2, corresponding to the same instruction set and same probability statistics) is compared with binary coding and Gray coding. As expected, Huffman code gives better results (less switching activity) than other encodings in all cases. This is true both for switching activity per clock cycle as well as switching activity per line per clock cycle (Note: Different encodings yield control path with different number of lines). However, the extent of reduction differs for different programs. This is because of different instruction mixes.

However, one issue that needs to be addressed is the dependence of hardware complexity on the encoding scheme. Our scheme is found to reduce the switching activity per line(Figures 5 and 7), but increase in number of lines can result in increased static power consumption. This is a tradeoff issue and will be studied in our future work.

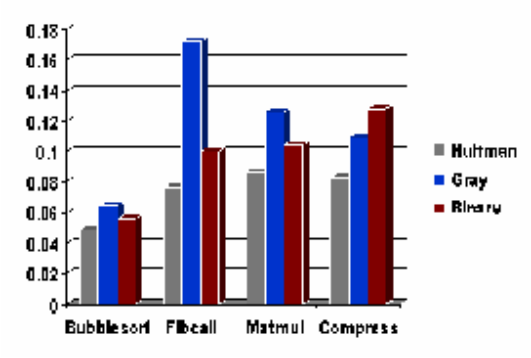

Fig 4. Switching Activity per Clock Cycle(Huffman Encoding 1)

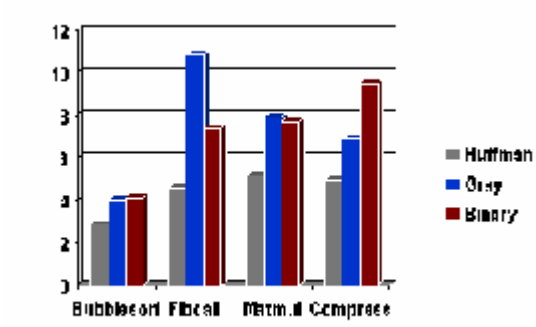

Fig 5. Switching Activity per Line per Clock Cycle(Huffman Encoding 1)

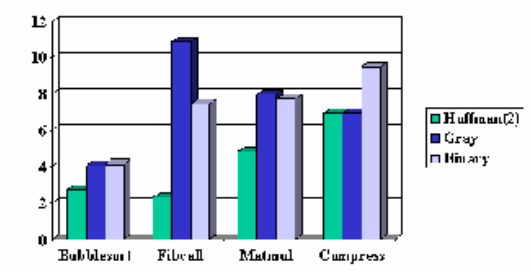

Fig 6. Switching Activity per Clock Cycle(Huffman Encoding 2)

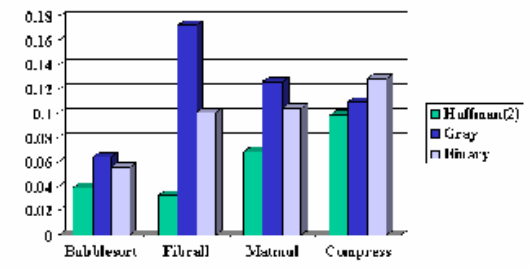

Fig 7. Switching Activity per Line per Clock Cycle(Huffman Encoding 2)

Bibliography

[1] S.Ramprasad, N.R. Shanbag, and I.N. Hajj, "Information-Theoretic Bounds on Average Signal Transition Activity," *IEEE Trans. VLSI Syst.*, vol.7, pp. 359-368, Sept. 1999.

[2] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, "Power estimation for sequential logic circuits", *IEEE Trans. VLSI Syst.*, vol. 3, Sep. 1995.

[3] A. P. Chandrakasan and R.W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp, 498-523.

[4] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE*, vol. 77, pp. 1879-1895, Dec.1989.

[5] S. Imam and M. Pedram, "An approach for multilevel logic optimization targeting low power," *IEEE Trans. Computer-Aided Design*, Vol. 15, pp. 889-901, Aug. 1996.

[6] A. Shen, A. Ghosh, S. Devdas, and K. Keutzer, " On average power dissipation and random pattern testability of CMOS combinational logic networks," in *Int. Conf. Computer-Aided Design*, Santa Clara,CA, Nov. 8-12,1992,pp. 402-407 .

[7] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp 473-484, Apr. 1992.

[8] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design", in *IEEE Symp. Low-Power Electron. Design*, San Diego, CA, Oct. 10-12, 1994, pp. 8-11.

[9] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic zero-transition activity encoding for address buses in low-power microprocessor-based systems," in *Great Lakes VLSI Symp.*, Urbana, IL, Mar. 13-15, 1997, pp 77-82.

[10] R. J. Fletcher, "Integrated circuits having outputs configured for reduced state changes," U.S. Patent 4 667 337, May 1987.

[11] S.Ramprasad, N.R. Shanbag, and I.N. Hajj, "A coding framework for low-power address and data busses," *IEEE Trans. VLSI Syst.*, vol.7, pp. 212-221, June 1999.

[12] M. R. Stan and W. P. Burleston, "Two-dimensional codes for low-power," in *Int. Symp. Low-Power Electron. Design*, Monterey,CA, Aug. 12-14, 1996, pp. 335-340.

[13]"Bus-invert coding for low-power I/O," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 49-58, Mar. 1995.

[14] C. L. Su, C. Y. Tsui, and A.M. Despain, "Saving power in the control path of embedded processors," *IEEE Design Test Comput. Mag.*, vol. 11, pp. 24-30, 1994.

[15] D. W. Faulkner, "PCM signal coding," U.S. Patent 5 062 152, Oct. 1991.

[16] http://www-rocq.inria.fr/scilab/ .