VALVE: Variable Length Value Encoder for Off-Chip Data Buses.

Dinesh C. Suresh, Banit Agrawal*, Walid A. Najjar and Jun Yang
Department of Computer Science and Engineering, University of California, Riverside
Department of Computer Science and Engineering, University of California, Santa Barbara
dinesh@cs.ucr.edu, banit@cs.ucsb.edu, {najjar, junyang}@cs.ucr.edu

Abstract

We propose VAriable Length Value Encoding *(VALVE) technique to reduce the power consumption in the off-chip data buses. While past research has focused on encoding fixed length data values to reduce the transition activity in the data buses, our proposed scheme is capable of detecting and encoding variable length bit patterns in the data values. VALVE also does not require prior knowledge of input data and uses just one external control signal*

We evaluate our proposed scheme using a large spectrum of benchmarks and we achieve an energy reduction of 58% on an average and up to 75% on some benchmarks. We also analyze the performance penalty incurred due to the codec delay, which is found to be 0.45% of the total program execution time. We find that VALVE requires a minimal area of 0.0486 mm^2, which can be easily implemented with in a memory controller.

1. Introduction

Power consumption in the bus drivers is in direct proportion to the product of the number of signal transitions, at each cycle, and the line capacitance. Bus encoding schemes are techniques that reduce the bus power consumption by reducing the number of transitions on the bus. The overhead associated with encoding bus values is found to be negligible compared to the energy saved during off-chip transmission [8] .

Frequent Value Encoding (FVE)[9] is a data bus encoding scheme that employs a k-bit, k-entry table to store previously seen data values, where k is the width of the data bus (32 in this case). During the first occurrence of a data value, the codec stores the data value in its table and sends the value unencoded. For subsequent occurrences of the data value, the codec sends a one-hot code instead of sending the entire data value. One-hot code denotes a value whose binary representation has a high value in only one of the bit positions.

A *full match* in a data bus encoding scheme is when all the k-bits of an incoming data value matches with the corresponding bits of the stored data value. In a *partial match* event only a small number of the k bits do not match. Our analysis has shown that the occurrence of partial match events is three times more frequent than that of full match events. We propose *VA*riable *L*ength *V*alue *E*ncoding (VALVE), a scheme capable of encoding both full matches and variable length partial matches in data streams. VALVE provides up to 75% reduction in energy for some applications and 58% energy savings on an average over unencoded data.

2. VALVE Design

VALVE uses Content Addressable Memories (CAMs) to store a finite set of table entries at the encoder and decoder ends. Each CAM table-entry consists of a variable-width bit pattern and a fixed width code. A VALVE table segment consists of a group of table-entries that store patterns of the same width. VALVE uniquely maps each bit incoming pattern to one of the available codes stored in the table. The bit-patterns are inserted into CAMs during the first occurrence of the databus value. For subsequent occurrences of the same value (or its portion), VALVE sends the corresponding unique code instead of sending the data.

VALVE asserts an external control signal whenever encoded values are sent on the bus wires. The fixed width code is always sent on a predetermined set of bus wires. In order to ensure integrity of the encoding/decoding operation, the width of the encoded portion should be greater than or equal to that of the code width. This ensures that the remaining bus wires are adequate to send the unencoded portion of the data.

In Figure 1, we show the block diagram of VALVE encoder that can encode bit patterns of width 32-bits, 24-bits and 16-bits. For every data value masks are applied to extract the 32, 24 and 16 bit patterns. These bit patterns are then looked up in the appropriate segments of the VALVE table. In the event of a hit in multiple segments, the segment selector picks the hit code from a segment with the largest segment-mask (code[hit_index] in *Figure 2*). The k-bit code from the hit location constitutes the upper k-bits of the current data bus value. The complement of the hit segment's mask is logically ANDed with the data value in order to get the low order bits of the encoded data bus value. The code is ORed with the low order data bits and the final 32-bit value is sent on the data bus..

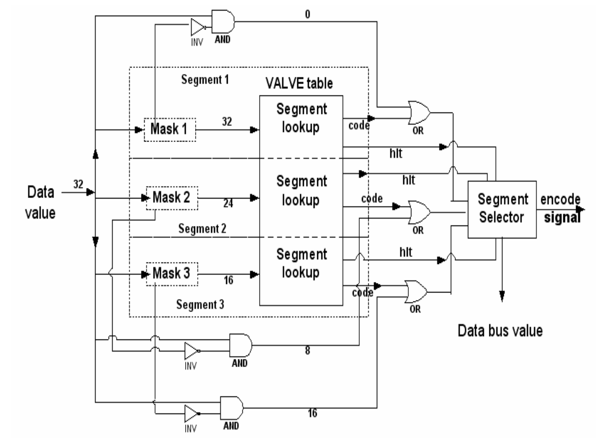

Figure 1 shows a VALVE encoder with three segments of 32, 24, and 16-bit width. The segment selector gives high priority to segments with larger bit width (32 in this case).

Figure 3 shows a VALVE decoder with three segments of widths 32-bits, 24-bits and 16-bits respectively. The upper *k-bits* of the data-bus-value contain the code if the data value is encoded. Upon receiving an encoded data bus value, the decoder searches for the upper *k-bits* in its VALVE tables. In the event of a hit, the VALVE table returns a data value associated with the search code. This returned data value constitutes the high-ordered bits of the decoded data value. The complement of the hit segment's mask is applied to the current data bus value in order to obtain the lower order bits of the decoded data value. When the external encode signal is low, the decoder interprets the valie "as-is".

3. Experimental Setup

We implemented our encoder in the *sim-outorder* simulator in the Simplescalar toolset [1]. Our test programs consisted of 23 benchmarks from the MediaBench [5], NetBench [6], MiBench[4], benchmark suites and four applications from SPECINT2000 [7].

VALVE encoder

```
1.  For each data value do
2.  If hit in valve table then
3.    encode signal = 1
4.    current data bus value = code[hit_index] OR
          (complement(mask) AND data value)
5.  else
6.      encode signal = 0
7.      current data bus value = data value
8.      insert-in-valve-table ( data value )
9.  end if
10. end for
```

Figure 2 shows our VALVE encoding algorithm. Each incoming data is encoded based on a hit in VALVE table; otherwise it is sent unencoded.

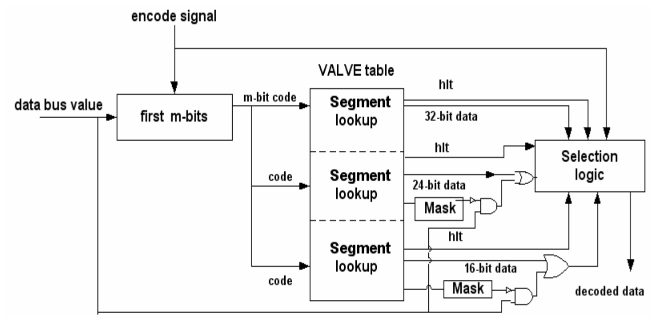

Figure 3 shows a VALVE decoder with three segments of 32, 24, and 16-bit width. The selection logic selects the proper segment and decodes the final 32-bit value.

For SPECINT applications, we fixed the L1 and L2 data cache sizes at 64KB and 512KB respectively. For rest of the embedded system applications, we evaluated L1 data caches of the following sizes – 1KB, 2KB, 4KB and 8KB. For each of these cache configurations, we fixed the block size, on-chip latency, off-chip latency and data bus width to be 32 bytes, 1 cycle, 100 cycles and 32-bits respectively. The off-chip data value in our simulation consists of both instruction and data values.

We use a bus power model similar to the one discussed by [2]. Although estimating the energy used in the off-chip interconnects is difficult, we can reasonably approximate the capacitance for the bus using the formula:

$$C_{bus} = C_{metal} * \text{No. of Bus lines}$$

In this expression C_{metal} is the capacitance of the metal interconnect for each bus line. Using the numbers given in [2], it is estimated to be 20pF. C_{bus} gives the effective capacitive load to be driven during a bus transaction. We calculate the total bus energy per cycle using the following formula:

$$E_{total} = E_{enc} + \frac{\{T_r * C_L * V^2\}}{\#\text{ of cycles}} + E_{dec}$$

where, T_r = *number of transitions in the off-chip bus*
C_L = *Load capacitance of the off-chip bus line.*
V = *Supply voltage.*

Parameters used for the calculation are: C_L = 20pF and V = 3.3 Volts.

4. Results and Discussion

We compare our scheme with frequent value encoding (FVE) scheme, which is known to work best among the existing data encoding schemes. Figure *4* shows the percentage reduction in energy for FVE and VALVE scheme while using an L1 data cache of size 4KB. The SPECINT applications shown in the graph used an L1 cache of size 64KB and an L2 cache of size 512KB. We find that for almost all the benchmarks, our scheme provides extra energy savings of 6% to 20% over FVE scheme. Pointer-intensive applications like *mcf* and *parser* have extensive

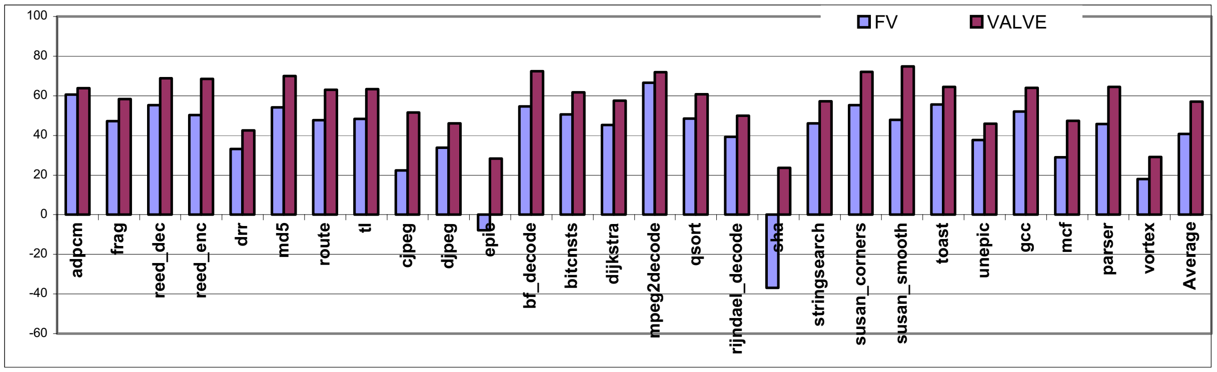

Figure 4 shows the percentage reduction in total energy (including codec's energy) for all applications. (L1cache of 4KB)

partial matches in the MSBs and hence, VALVE offers nearly 20% energy improvement over FVE for such programs. While in some of the benchmark such as *epic* and *sha,* FVE performs worst than the unencoded data; our scheme provides an energy saving of more than 20%. Table 1 shows the percentage reduction in energy for different cache configurations. As we can see from the table that, as we increase the cache size, the energy saving reduces. This may be due to reduction in the number of repeated bit-patterns. On an average, the partial data segments accounted for 73% of the total segment hits.

On average, VALVE incurs 0.06%, 0.29%, and 0.76%, performance penalty with codec delay of 2 cycles for MiBench, MediaBench, and NetBench respectively. We evaluated the area requirement for VALVE codec and found that it could be implemented with a minimal area of 0.0486 mm^2.

5. Conclusion

VAriable Length Value Encoding (VALVE) technique minimizes the energy consumption in the off-chip data buses by encoding and decoding variable width repeated bit-patterns. We measure the energy consumed by the codec and evaluate the reduction in off-chip bus energy. VALVE provides up to 75% reduction in energy for some applications and it yields 58% reduction in energy on an average, whereas it provides an average energy savings of 16% over FVE scheme. VALVE provides this significant energy savings at the expense of a very small performance overhead. We find that our codec can be implemented with a minimal area of 0.0486 mm^2.

Table 1 shows the average percentage reduction in energy for different cache configurations.

Cache size	FVE	VALVE
1KB	44.94	59.50
2KB	40.88	58.89
4KB	40.87	57.14
8KB	40.62	56.53

References

[1]. D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0, Technical Report", University of Wisconsin-Madison, Computer Science Department, 1997.

[2]. F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele and A. Vandecappelle, " Exploration of Memory Organization for Embedded Multimedia System Design", *Kluwer Academic Publishers",* 1998.

[3]. J.H. Chern, J. Jurang, L. Arledge, P. Li and P. Yang, "Multi-level Metal Capacitance Models for CAD Design", *IEEE Electron Device Letters*, Vol13, pp 32-34, Jan. 1992.

[4]. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. Austin, T. Mudge and R. B. Brown, " MiBench: A Free, Commercially Representative Embedded Benchmark Suite", IEEE 4th Ann. Workshop on Workload Characterization, Austin, TX, Dec. 2001

[5]. C. Lee, M. Potkonjak and W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", *Intl. Symp. on Microarchitecture*, pages 330-335, 1997.

[6]. G. Memik, W. H. Mangione Smith and W. Hu, "NetBench : A Benchmarking Suite for Network Processors", *Intl. Conf. on Computer Aided Design (ICCAD)*, pp 39-42,San Jose, California, Nov. 2001.

[7]. SPECINT2000, http://www.specbenh.org/cpu2000

[8]. D. C. Suresh, B. Agrawal, J. Yang, W. Najjar and L. Bhuyan, "Power Efficient Encoding Techniques for Off-Chip Data Buses", In the Proc. of Compilers and Architecture and Synthesis for Embedded Systems (CASES), San Jose, CA, Oct. 2003

[9]. J. Yang and R. Gupta, "FV-Encoding for Low Power Data I/O", ACM/IEEE Intl. Symp. on Low Power Electronic Design", Pages 84-87, 2001.