A Power-Configurable Bus for Embedded Systems

Chuanjun Zhang
Dept. of Electrical Engineering
University of California, Riverside
czhang@ee.ucr.edu

Frank Vahid
Dept. of Computer Science & Engineering
University of California, Riverside
vahid @cs.ucr.edu, www.cs.ucr.edu/~vahid
Also with the Center for Embedded Computer Systems at UC Irvine

Abstract

Pre-designed configurable platforms, possessing microprocessors, memories, and numerous peripherals on a single chip, are increasing in popularity in embedded system design. Platform configurability enables use in more products, which results in lower cost platforms due to higher volume production. Common configurable features include voltage scaling and cache organization. We introduce a new bus architecture that can be configured for normal performance or low power. The additional hardware that provides this configurability imposes almost no performance overhead in normal mode compared to a non-configurable bus. We show the substantial range of power and performances that such a bus provides, and we show that the low-power configuration can result in energy savings, using a set of benchmarks.

Keywords

Bus, low power, configurable systems, system-on-a-chip, platforms, architecture tuning.

1. Introduction

Platforms are becoming popular in embedded system design. A platform is an off-the-shelf system-on-a-chip (SOC) typically consisting of a microprocessor, cache, memories, coprocessors, peripherals, and a hierarchy of buses [13]. A platform is typically targeted to a particular application domain, such as network switches, digital cameras, set-top boxes, or portable audio devices. While some platforms are intended for prototyping only, others are intended for use in final products.

Embedded system designers gain two key advantages by using an off-the-shelf platform integrated circuit (IC) in a product compared with developing a semi-custom IC. First, they reduce time-to-market, both by performing extensive in-circuit verification and thus eliminating costly errors early, and by eliminating the months required for IC design and fabrication. Second, they may achieve low product cost due to the low platform cost, which is possible due to the platform manufacturer selling the platform in high volumes to designers of a variety of products. Use of the same platform in a variety of products enables the platform designer to amortize non-recurring engineering costs over larger volumes than possible for a semi-custom IC intended for a single product.

For a platform to be usable in a variety of products, it must be configurable to the varying power and performance demands of different products [11]. In the same application domain, one product may require higher performance, while another may require lower power. In fact, even the same product may have changing requirements over time – high performance when plugged in to a power supply, low power when running on battery, for example.

Thus, a need exists for new architectural components that can be configured for either high performance or low power. In this paper, we highlight previous work on components that have appeared in recent products or been proposed in research. We then introduce a new bus architecture that can be configured to execute in normal (high performance) mode, or a slower low power mode. We provide power and performance data for several examples from the Powerstone benchmark suite, showing a good range of power and performance. We also provide energy data demonstrating that the bus can reduce the total energy required for many examples.

2. Related Work

Voltage scalable microprocessors represent one form of performance/power configurable component appearing in several recent devices [5]. Such microprocessors can have their source voltage reduced, with a corresponding reduction in clock frequency, to reduce power. Since much power consumption in CMOS circuits is proportional to the square of the source voltage [12], power reductions can be huge, with a single processor having active operation power consumption ranging from a few Watts down to just milliwatts.

Figure 1: Base architecture**.**

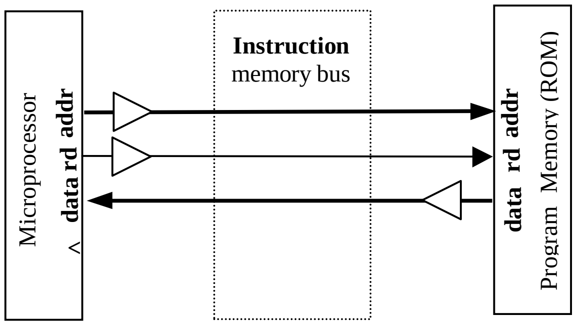
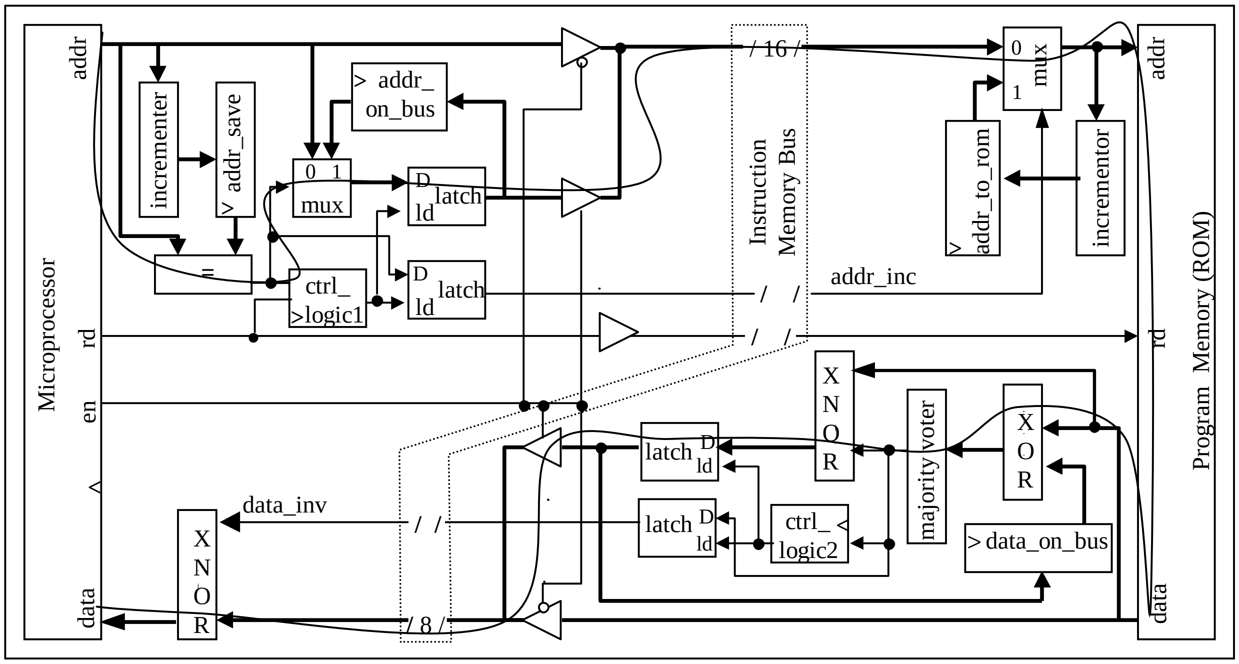


Figure 2: Power-configurable bus architecture**.** The critical path when encoding is activated is shown.



Configurable caches are also beginning to appear. Adding an on-chip cache is a well-known method for improving microprocessor performance. An N-way set-associative cache (where common values for N range from 2 to 32) is a popular cache design. However, because the N ways must be simultaneously searched during each cache access, N-way caches can consume a large percentage of system power. Thus, caches that can have some of their ways shut down have recently been proposed [1] and have appeared in commercial products [7]. For some applications that didn't really need all the ways, the impact of shutting down some ways on misses (and hence performance) may be very small, resulting in less power per cache access with little performance loss. For other applications, the impact on misses may be greater, but the power savings per cache access may still outweigh the extra power consumed by the additional transfers between memory and cache due to misses. Thus, power is reduced at the expense of performance. For yet other applications, however, the extra miss power outweighs the cache access power savings, so for those examples, shutting down ways actually increases power. Thus, determining the number of ways to shut down when lower power is needed is highly dependent on the application.

The voltage scaling and cache way shutdown methods can obviously be combined in a single platform. At the lowest voltage setting, one might shut down the optimal number of ways for a given application to further reduce power. For the highest voltage setting, one might enable all ways to give best performance. For voltage levels in between the highest and the lowest, one might use the configurable ways to provide more power/performance points, resulting in finer-grained tradeoffs.

3. A Power-Configurable Bus Architecture

3.1 Overview

While voltage scaling provides tremendous ability to tradeoff performance and power, more can be done. Without additional configurable architectural components, we have the rather odd situation of a system whose voltage can be scaled down to reduce power, but whose cache, bus and other architectural components are still configured as if high performance were the goal. Furthermore, voltage scaling requires fairly costly DC-to-DC converters, accompanied by clock frequency scaling circuitry, so may not be feasible in all platforms. Configurable architectural components, like a cache whose ways can be shut down, can be used to address these problems.

We introduce a configurable bus, which can also be used to tradeoff power and performance. Buses can consume much power in an SOC, which may be as high as 80% of the total power dissipation [9]. This is especially true as features continue to shrink, making wires even more costly [13].

There are three components in CMOS power consumption: dynamic power consumption, short circuit power consumption, and static power consumption. In many cases, the dynamic consumption is dominant. The dynamic power consumption in a CMOS circuit can be described by the equation $P_d = C_l V^2_{dd} f_p$,, where f_p is the switching frequency, C_l is the average capacitance loading of the circuit, and V_{dd} is the supply voltage. Our configurable bus seeks to reduce the switching frequency. Other techniques, such as those that reduce voltage, can additionally be applied to reduce power further.

We introduce our configurable bus between a microprocessor and a program memory. However, the same concepts can be applied to other buses, such as those between microprocessor and cache, between cache and memory, and even to buses connecting coprocessors or peripherals.

In our base architecture, shown in Figure 1, an Intel 8051 microprocessor is connected with a program memory (ROM) via an instruction memory bus. This bus consists of 16 address lines (*addr*), 8 data lines (*data*), a read line (*read*), and a clock line (*clk*).

We extend this base architecture as shown in Figure 2. There are two main extensions. At the top of the figure is logic to encode (on the processor side) and decode (on the memory side) addresses. At the bottom of the figure is the logic to encode (on the memory side) and decode (on the processor side) data. The goal of encoding is to transmit the same information while minimizing bit switching on the buses. The key assumption is that switching on a bus wire consumes far more power, perhaps 100 to 1000 times, than switching of internal logic wires [10].

3.2 Data Encoding

For the data bus, we have chosen to use bus invert encoding [10]. The idea of this approach is to compare the current data to be transmitted with the previously transmitted data. If more bits would switch than would stay the same, we instead transmit the complement of the current data, plus an additional bit indicating that the data has been complemented.

In Figure 2, the *data_on_bus* latch stores the previously transmitted data. When current data comes from the ROM, it is compared with the previous data through 8 *XOR* gates that each compare 1 bit position of the current and previous data. The *majority_voter* circuit outputs a one if its input has more ones than zeros, which would mean more bits are different than are the same. In this case, the complement of the current data is obtained using an XNOR gate, the *data_inv* signal is set to one, and the data and *data_inv* signals are stored in latches that drive the data bus, all under the control of the *ctrl_logic2* control logic block. If the *majority_voter* circuit outputs a zero, then the true value of data is stored in the bus latch, and *data_inv* is set to zero. The latches that drive the data bus are necessary to prevent glitches, which occur as the encoding logic computes, from appearing on the bus. They do not use the clock and thus do not add an extra clock cycle.

On the microprocessor side, we simply feed the data through 8 XNOR gates that each compare 1 bit of the data with the *data_inv* signal. If *data_inv* is zero, data propagates unchanged. If *data_inv* is one, the data gets complemented.

3.3 Address Encoding

For addresses, we have chosen to use T0 encoding [3]. Although we could have chosen to use bus-invert for addresses also, T0 can do better when the values appearing on the bus are mostly sequential (incremented by one over the previous value). This is indeed the case for instruction bus addresses. The idea of T0 encoding is to compare the current address with the previous address. If the current address is the previous address plus one, we hold the bus steady and instead set an additional line that tells the memory to increment the previous address.

In Figure 2, the previous address plus one is stored in *addr_save*. The current address is compared with this incremented value. If equal, then the current address equals the previous address plus one. In this case, the previous bus value, stored in *addr_on_bus*, passes through a multiplexor and is stored in the latch that drives the address bus. This prevents the bus from switching. Meanwhile, the control logic *ctrl_logic1* stores the one coming from the comparator into the latch that drives the *addr_inc* line. If the current address was not equal to the previous address plus one, then the current address is passed through the mux into the latch that drives the address bus, and the *addr_inc* latch is set to zero.

On the memory side, if the *addr_inc* line is zero, the address bus passes through a mux to the ROM. If the *addr_inc* line is one, then the previous value fed to the ROM plus one, stored in register *addr_to_rom* on every clock cycle, is passed through the mux to the ROM.

3.4 Configuration Logic

All of the encoding and decoding logic we have described is enabled by the signal *enable* coming from the microprocessor. This signal can be software programmable, or alternatively can be directly connected to an external pin on the IC.

When *enable* is zero, the tri-state drivers at the top and bottom of the center of Figure 2 pass the data and address signals unchanged. Notice that when enable is zero, the signals pass through no logic other than the tri-state drivers. As the base architecture had bus drivers too, the extra delay imposed by using tri-state drivers is very small if any.

When enable is one, the tri-state drivers pass the encoded values onto the buses. For encoding, the signals must pass through additional logic. The additional logic that must be passed through for a complete instruction fetch is shown in Figure 2. Note that none of the additional logic is clocked. In other words, we have designed the encoding logic to ensure that we do not introduce additional clock cycles, since this could require expensive changes to the microprocessor control logic.

Instead, we have lengthened the instruction fetch time. If instruction fetch time represented the system's critical path (as it often does), then enabling encoding lengthens the critical path. This in turn means that the system clock frequency must be reduced for proper operation when encoding is enabled.

While Figure 2 graphically shows the encoding/decoding logic as quite large, this does not reflect the physical design. In particular, the bus lines themselves may be very large. In our case, the bus lines connect to an in-system programmable ROM, which is separated from the microprocessor by a long distance. Those wires are thus long and wide. In other words, the wires internal to the encoding/decoding logic will be much shorter and narrower than the bus wires. Also recall that the transistors and wires related to encoding/decoding logic consume far less power than the long bus wires.

4. Experiments

We have implemented our configurable bus as the instruction bus of a system consisting of an Intel 8051 microcontroller interfaced with a ROM program memory. We modeled the entire system as a gate-level VHDL file. The 8051 was synthesized from the model available at [6].

We performed simulations and power analysis of numerous benchmarks executing on the 8051 using Synopsys tools. The benchmarks were examples taken from the Powerstone benchmark suite [8]. *Binary* is a binary search for the array of 15 integer elements; *Insert* is a insertion sort for 10 integer

Figure 3: Power results**.**

Benchmark	addr power (mW)			data power (mW)			total power (mW)			cycles	cycle (ns)		total energy (mJ)		
	!en	en	%	!en	en	%	!en	en	%		!en	en	!en	en	%
Binary	0,50	0,15	71%	1,27	0,66	48%	2,12	1,46	31%	17.274	50	60	1,83	1,51	18%
Insert	0,38	0,15	59%	0,71	0,55	22%	1,90	1,40	26%	53.366	50	60	5,06	4,47	12%
Matmul	0,43	0,05	89%	0,75	0,48	36%	2,05	1,31	36%	507.536	50	60	51,92	39,80	23%
Ucbqsort	0,44	0,08	82%	0,62	0,46	27%	1,98	1,36	31%	260.000	50	60	25,71	21,14	18%
Blit	0,73	0,02	97%	0,98	0,36	63%	2,15	1,06	50%	20.895.620	50	60	2243,14	1333,98	41%
Brev	0,35	0,08	78%	0,70	0,46	35%	1,82	1,07	41%	1.402.772	50	60	127,93	90,31	29%
Average	0,47	0,09	79%	0,84	0,49	38%	2,00	1,28	36%	3.856.095	50	60	386,00	294,99	23%

en = encoding enabled

numbers; *Matmul* is a matrix multiplication for 5x5 integer matrices; *Ucbqsort* performs quick sort, and *Blit* is a graphics application. The power we report includes the power on the bus and the encoding/decoding logic. We use a bus capacitance to internal wire capacitance ratio of 1000:1 [2].

The critical path of the 8051/memory system without encoding was 50 ns. The encoding/decoding logic when activated added 10 ns to this path, making the path 60 ns. In other words, without encoding, the fastest system clock frequency is 20 MHz. With encoding, the frequency must be reduced to 16.7 MHz.

The results are summarized in Figure 3. The power due to the address bus and data bus, with encoding disabled (*!en*) and enabled (*en*), are shown separately. Address bus power savings ranged from 59% to 97%, with the variation being due to some applications have more sequential address access than others. Likewise, data bus power reductions ranged from 22% to 63%, with the variation due to different applications having differing correlations between successive data items. Total system power reductions, which now include the power consumed by the microprocessor and memory plus the bus encoding/decoding logic, ranged from 26% to 50%, averaging 36%.

The energy required to perform a computation is another metric of interest, equal to power times time. Note that reducing power does not necessarily imply energy reduction, because the time increase could outweigh the power decrease. However, in our architecture, energy savings do occur, ranging from 12% to 41%, and averaging 23% across all the examples.

5. Conclusions

We have presented a bus architecture that can be configured for regular performance or for low power. The architecture uses encoding logic that can be enabled to reduce switching on the bus and hence reducing power, at the expense of performance. The design is such that the performance and power overhead when the encoding logic is disabled is minimal. Such a bus can be used in a platform along with other configurable features, such as voltage scaling or cache configuration, to provide for power and performance tradeoffs. Our future plans include integrating this bus with configurable cache and loop cache components [4].

6. Acknowledgments

This work was supported by the National Science Foundation (grant #CCR-9876006).

References

[1] D.H. Albonesi, "Selective cache ways: on-demand cache resource allocation," Journal of Instruction-Level Parallelism 2(2000) 1-6.

[2] H.B.Bakoglu, Circuits, Interconnections, and Packaging for VLSI.Addison-Wesley 1990.

[3] L.Benini, G. De Micheli, E. Macii, D.Scjuto, C.Silvano. "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems," GLS-VLSI-97:IEEE 7th Great Lake Symposium on VLSI.pp.77-82:Urbana-Champaign:IL:March 1997.

[4] A. Gordon-Ross, S. Cotterell and F. Vahid. Exploiting Fixed Programs in Embedded Systems: A Loop Cache Example. IEEE Computer Architecture Letters, Jan. 2002.

[5] http://developer.intel.com/design/intelxscale/benchmarks.htm.

[6] The UCR Dalton Project, http://www.cs.ucr.edu/~dalton.

[7] A. Malik, B. Moyer and D. Cermak. "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility." International Symposium on Low Power Electronics and Design. June. 2000.

[8] A. Malik, B.Moyer, and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility". Proceedings of ISLPED2000,Rapallo, Italy, 26-27 July 2000.

[9] C. A. Neugebauer, R. O. Carlson, "Comparison of Wafer Scale Integration with VLSI Packaging Approaches, " IEEE Transactions on Components, Hybrids, and Manufacturing Technology, pp. 184-189, June 1987.

[10] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 1, pp. 49-58, March 1995

[11] F. Vahid and T. Givargis. Platform Tuning for Embedded Systems Design. IEEE Computer, Vol. 34, No. 3, pp. 112-114, March 2001.

[12] N.H.E. Weste and K.Eshraghian, "Principles of CMOS VLSI Design, A System Perspective" Second Edition. Addison-Wesley. 1993 pp. 233-235.

[13] Semiconductor Industry Association. International Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: International SEMATECH, 1999.