

Bus encoding architecture for low-power implementation of an AMBA-based SoC platform

S. Osborne, A.T. Erdogan, T. Arslan and D. Robinson

Abstract: Advanced microcontroller bus architecture (AMBA) is rapidly becoming the *de facto* standard for new system-on-chip (SoC) designs. The bus protocol is complex, making any peripherals that can interface to it valuable intellectual property (IP). This paper presents a low-power bus encoding architecture which is able to deal with the complex advanced high-performance bus (AHB) protocol within AMBA, which involves multiple burst transfers. The architecture is targeted for a low-power SoC platform to be used in a miniaturised low power application area. The paper describes the SoC platform and the bus encoding architecture, and provides results with a design synthesised at 0.35 μm CMOS technology indicating up to 22% power saving.

1 Introduction

For CMOS circuits, most dynamic power dissipated is for charging and discharging node capacitances. Dynamic power dissipated by a CMOS circuit is of the form [1]:

$$P_{\text{chip}} \propto \sum_{i=1}^N C_{\text{load}_i} * V_{dd}^2 * f * pt_i \quad (1)$$

The summation is carried out over all N nodes in the circuit, C_{load_i} is the load capacitance at load i , V_{dd} is the power supply voltage, f is the frequency and pt_i is the activity factor at node i .

In a CMOS VLSI circuit that uses well designed gates, the switching activity accounts for over 90% of the total power consumption. In circuits optimised for low power, the total power dissipation at the I/O is typically around half of the total for the circuit. This is due to the significantly larger dimensions of the devices in the I/O pads required to drive the external capacitances and also the external capacitance from the I/O pins, wires and peripheral circuits. To produce a clearer model of the impact of I/O transitions, (1) can be simplified by assuming all nodes to have the same average load capacitance, and V_{dd} and f to be the same for all nodes [1]:

$$P_{\text{chip}} \propto C_{\text{average}} * V_{dd}^2 * f * N(\text{transitions}) \quad (2)$$

In fact, this is oversimplifying the situation as a huge variation exists between average node capacitance for on chip and I/O nodes. The capacitance associated with an

external pin is typically larger than that of an internal node by 2–3 orders of magnitude. The model must, therefore, be further shaped to take account of this, as shown in the following [1]:

$$P_{\text{chip}} \propto C_{\text{int}} * N(\text{transitions})_{\text{int}} + C_{\text{ext}} * N(\text{transitions})_{\text{I/O}} \quad (3)$$

A clear model now exists for the total power dissipated by the chip being contributed by the internal and I/O factors. The number of internal transitions is generally much larger due to the greater number of nodes, but the load capacitance will be several orders of magnitude smaller.

From this theory, the idea of coding can therefore be justified to decrease the number of transitions on the large capacitance side, even at the expense of slightly increasing the number of transitions in the internal circuit. Intuitively, the total power dissipation should decrease by decreasing the number of transitions on the high-capacitance side, as the number of internal transitions is already large and increasing it slightly is unlikely to be significant.

2 Bus coding techniques

In general, bus encoding techniques aim to reduce the power consumption on a bus by mapping the information conveyed on the bus to a form which has less transition activity than the original. These are largely based on the fact that the relationships between the data can be developed and exploited, between subsequent transmissions, to reduce the switching activity on the lines. This work focuses on off chip buslines, due to their high capacitive load which makes them a major contributing factor to the overall power consumption of the SoC device.

Various low-power coding methods have been proposed in the literature for both address and data bus. For the address bus, coding techniques such as Gray [2], zero-transition activity (T0) [3], T0-XOR [4], offset [4] and pyramid code [5] have been used effectively. For data bus encoding, on the other hand techniques such as bus-invert [1, 6], transition signalling [6], BITS [7], codebook-based encoding [8] and probability-based mapping [9] have been investigated. In specific applications, when the characteristics of the used data can be exploited, techniques such as the Beach solution

[10] and partial bus-invert [11] have proved successful. Others, such as working-zone [12] and entropy-reducing code [13], have been used for both data and address bus coding.

This work targets the development of a bus encoding architecture for a generic SoC platform targeting a wide range of applications. For this reason, a general bus encoding technique is sought. It has been demonstrated in the literature that the bus-invert technique is the most suited for such generic applications where data are random, i.e. where *a priori* information about the stream that is transmitted is not required [9, 13]. In addition, its implementation leads to much less complex hardware for both decoding and encoding logic. The work focuses on reducing power on the data bus only as the nature of transitions on the address bus is completely different and its development for an SoC platform requires a dedicated special study, and hence, will be beyond the scope of this paper.

When using bus-invert for data coding an extra bit, known as the invert, is required. When the invert bit is set to 0 the value of the data bus is equal to the original data, and when it is set to 1 the data on the bus is equal to the inverse of the original data. Detailed operation of the basic invert scheme is described in [1]. Although it does require additional logic and bus signals, the additional I/O signals should be compensated by the reduced ground bounce, meaning less supply and ground pins are needed. This additional logic, however, is less complex than that required for alternative techniques such as the working-zone.

3 Implementation

3.1 SoC platform and external memory controller architecture

The architecture of the SoC platform for which this research is carried out contains much of the basic functionality

needed for a processor-based system. Considering Fig. 1, the platform consists of the following:

- AMBA advanced high-performance bus (AHB) and advanced peripheral bus (APB)
- $8\text{ k} \times 32$ internal memory
- an external memory controller
- two timers
- a UART
- an interrupt controller
- a system controller
- a system watchdog
- general purpose I/O (GPIO)
- ten expansion ports (three AHB masters, three AHB slaves, four APB slaves).

The AMBA AHB [14] is a new generation of AMBA bus intended to address the requirements of high-performance synthesisable designs. It is a high-performance system backbone bus on which the CPU and other direct memory access (DMA) devices reside. AMBA AHB sits above the APB and is able to sustain the external memory bandwidth, and implement the features required for high-performance, high clock-frequency systems. These features include: burst transfers, split transactions, single-cycle bus master hand-over, single clock edge operation, nontristate implementation, and wider data bus configurations (64/128 bits).

The external memory controller (EMC) sits on the AMBA AHB acting as a slave, controlling read and write access requests to the external memory. For this reason, the EMC is the primary candidate for the bus-invert coding/decoding logic. The interface to the AHB bus is clearly defined by the AMBA AHB protocol, and in modifying the EMC its functionality must not be violated. The interface of the EMC and its constituent modules are shown in Fig. 2. There are several off-chip signals, going to and from the external memory, but the signals on which the low-power techniques will be focused will be the data lines.

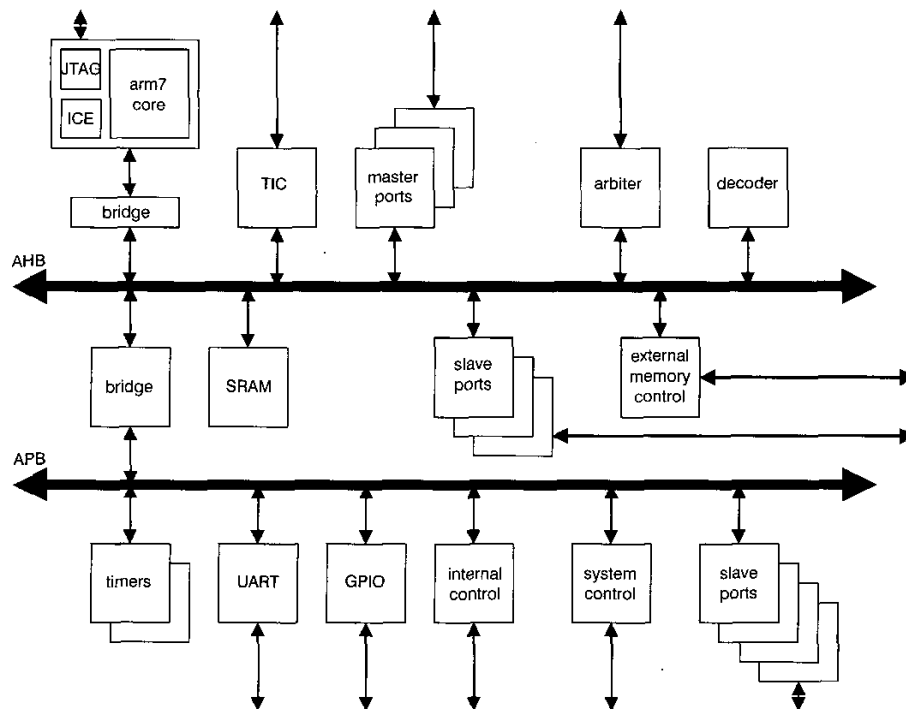


Fig. 1 Architecture of the generic SoC platform

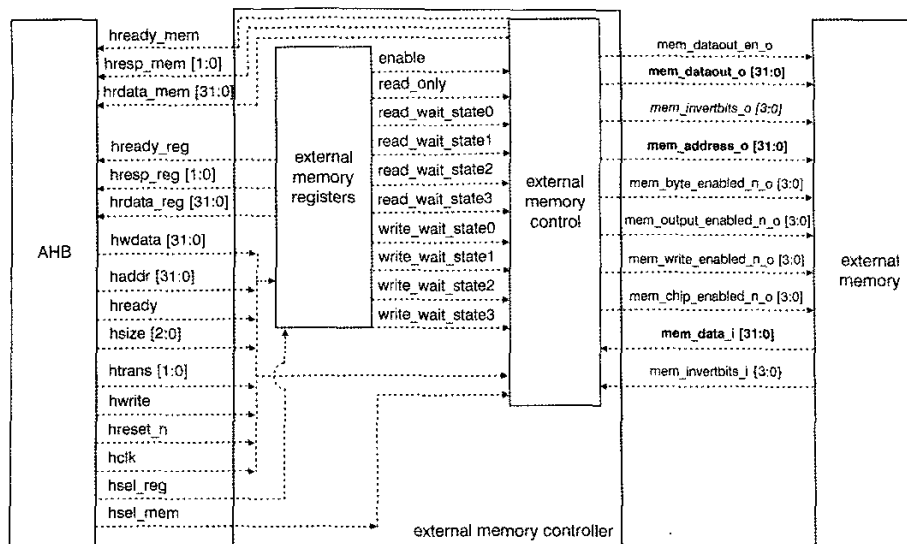


Fig. 2 High-level interface between the AHB and the external memory

Specifically, the connections of interest in reducing the switching power on external wires are 'mem_dataout_o' and 'mem_data_i'. These are the buses with the largest capacitance loading, using 32 bit lines each.

As shown by Fig. 2, the functionality of the EMC has been clearly split into two modules, which are then encapsulated by the top-level module. These modules are 'ahb_external_memory_registers', and 'ahb_external_memory_controller'. Both of the modules sit as slaves on the AHB, with 'ahb_external_memory_registers' acting as one slave, and 'ahb_external_memory_controller' effectively behaving as four separate slaves, controlled by 'hsel_mem'. Using 'hsel_mem' in this way removes the need to have multiple copies of 'ahb_external_memory_controller' module, and only uses one AHB slave port. Another way of configuring the system would be to configure 'ahb_external_memory_registers' as an APB slave, leaving another AHB slave free. This would be possible, as this module does not necessitate the high performance of the AHB.

3.2 Implementation of bus encoding and decoding for the AMBA protocol

This work involves the development of a specialised bus-invert scheme which is able to deal with the complex bus protocol associated with the AMBA and its integration within an SoC platform. The key issues which need to be tackled here are the development of specialised write and read coding mechanisms. The following two subsections will consider these issues respectively.

3.2.1 Write path encoding: The data received from the AHB bus is to be passed through a bus-invert coder function before being passed to the external memory. The activity on a typical data bus can be considered to be characterised by a random uniformly distributed sequence of values. With this assumption, at any given point, the data on an n -bit bus can be any of 2^n possible values with equal probability. The average number of transitions between transfers will therefore be $n/2$, and the maximum will be n . The bus-invert coding is optimal, when used with uncorrelated data, in the sense that, given the same redundancy, significant reduction in the number of transitions

can be achieved compared to others [13, 15, 16]. It therefore produces a perfect limited weight code [16]. However, for larger bus a partitioned bus-invert code may provide a better solution, but it will require extra lines. As the width (n) of the bus increases, the term $n/2$ becomes dominant and the decrease in average power is even smaller. To reduce the average power dissipation for larger bus, the observation is that the bus-invert coding performs better for small n suggests that the bus should be partitioned into several smaller bus.

In some applications, the system will be required to perform multiple byte addressing. This provides more flexibility to the system. This flexibility was needed in our target application area. Therefore, in this work the bus was split into four sections in order to perform 8, 16 or 32 bit transfers. For this reason, an invert bit was used for every 8 bits across the 32 bit data bus. Hence, the bus is increased by 4 bits to 36 bits. The drawback of this approach is that it will require 4 extra buslines to the external memory. This should not, however, make the coding any more difficult to implement, as the code will be expected to be able to cope with 8 bit, 16 bit and 32 bit addressable memories. Also, during write transfers, data comes off 'hwdata' on the appropriate byte lanes, and is fed to the memory to avoid overwriting any bytes not addressed. Therefore, correlating the 'mem_dataout_o' bus on a byte by byte basis should not present any problems, other than ensuring the invertbits are used consistently.

3.2.2 Read path decoding: The data received from the external memory is to be passed through a bus-invert decoder, which is simply a conditional invert logic based on the value of 'mem_invertbits_i', before being passed to the AHB bus. It is important to ensure that the correct invert bits are associated with the correct data bytes. This, however, is more of an issue when setting up the system environment and memory instantiations. No extra internal variable is required in this instance as the data is either passed unaltered from the memory to the AHB via the EMC, or it is inverted in the EMC before being passed to the AHB. These operations are again performed on a byte by byte basis. During read transfers, it is desirable for the external memory to return data only on the active lanes in

order to reduce switching power. Once the data is in the EMC it is then loaded into the correct byte lanes on 'hrdata'.

3.2.3 System integration: The implementation described so far has only focused on the EMC unit alone. However, in order to confirm the functionality of the EMC and bus-invert units the surrounding system must be established. A decision had to be taken on how to map the invert bits to the memory. Earlier in the integration process, the decision had been taken to separate the data and invert bit output signals. However, this does not restrict the memory mapping employed. The options were to either store the invert bits in the same memory as the data, requiring a 36 bit memory, or to instantiate new memories for the invert bits. As 36 bit memory models are not as freely available as 32 bit, it was decided to employ two memory modules and control these using the same address lines. The first is to store the data and the second to store the invert bits. This configuration is illustrated in Fig. 3.

4 Simulations and results

To enable the power analysis to be performed at gate level, the RTL code was synthesised to a specific target technology. The technology library selected was the Alcatel 0.35 μm library. This produced a netlist on which to perform a gate-level power analysis of the circuit. Power analysis was performed using Synopsys Design Power. The choice of the tool was made taking into the consideration the size of the design and the comparative nature of the research work carried out. Only the external memory controller was synthesised, as only simulation behavioural models were available for the memory and AHB structures. This would not be a problem at the EMC to AHB interface. However, consideration of the EMC to external memory interface was essential, as these lines are off-chip and hence highly loaded. To consider these external loads in the power calculations, it was necessary to include them somehow in the synthesis performed. This was achieved by using a wrapper to attach load buffers to the lines between the EMC and the external memory. This enabled the power analysis to consider transitions occurring on the off-chip buss to the external memory. The wrapper was created as a new top level module taking the output from each bit of each signal, attaching it to an internal wire in the wrapper, then passing it through a buffer, and out of the wrapper. The process is shown in Fig. 4 for a single bit of 'mem_dataout' and 'mem_datain' signals.

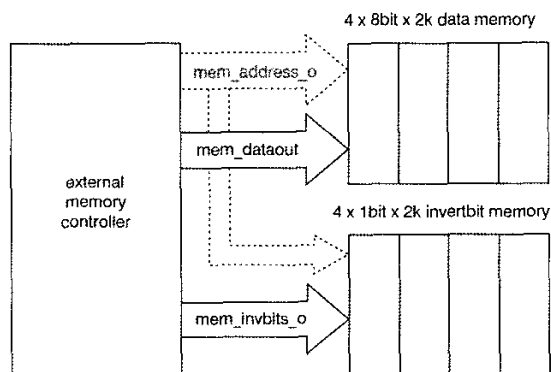


Fig. 3 EMC and memory configuration

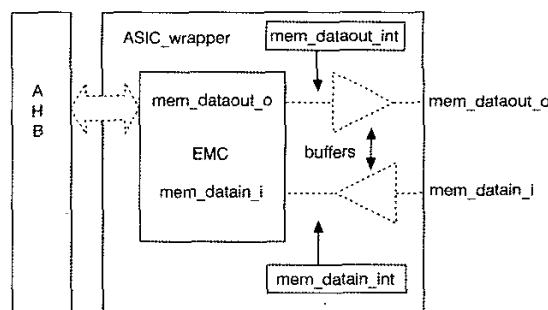


Fig. 4 Connections of EMC wrapper

Specific buffers were selected for input and output lines. This was done to make the wrapper more reusable as the buffers on all lines can be changed by simply altering the two input and output buffers instantiated in the wrapper.

Netlist simulations were conducted using 1000 memory write and reads to establish the impact of the bus-invert coding in the power consumption of the system. For these simulations randomly generated data were used. This case simulates the situations where the data are dynamic and not predictable. Tables 1–3 indicate the raw switching activity values for 8, 16, and 32-bit bus transfers, respectively. In all cases, the number of transitions introduced by bus-invert logic together with the actual data transitions are significantly less than the case where bus-invert is not implemented.

The power consumption results are shown in Fig. 5. The Figure indicates a significant saving of about 19% for a transfer size of 32 bits. The savings decrease to 5% and 1% for 16-bit and 8-bit transfer sizes, respectively. This agrees with the authors' observations because there is a wider range of activity in a larger transfer size. The analysis was repeated with a two-dimensional convolution algorithm used in image processing [17]. The power consumption savings profile was similar to the case shown here where

Table 1: Number of transitions for 8-bit transfers

Bus-invert	Low power off	Low power on
Data transitions	6022	4506
Invertbit transitions	N/A	240

N/A = not applicable

Table 2: Number of transitions for 16-bit transfers

Bus-invert	Low power off	Low power on
Data transitions	14876	11596
Invertbit transitions	N/A	1125

N/A = not applicable

Table 3: Number of transitions for 32-bit transfers

Bus-invert	Low power off	Low power on
Data transitions	28122	11558
Invertbit transitions	N/A	2166

N/A = not applicable

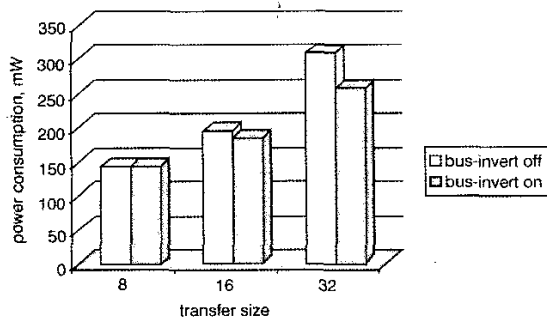


Fig. 5 Power consumption comparisons for bus-invert coding

Table 4: Low power logic area analysis

Configuration	Area	Area percentage change
EMC	68087	
EMC with our bus-invert logic	69343	+ 1.84%

random data were used. In this case, more than 22% savings were achieved with a 32-bit transfer size.

The area impact of our AMBA based bus-invert logic on the EMC could be seen in Table 4. The area is measured in number of equivalent two-input NAND gates after synthesis to the Alcatel 0.35 μm CMOS technology. The bus-invert logic has a small impact on the area, increasing it by less than 2%. However, it has been shown that this can provide more than 22% reduction in power overall with 32-bit transfers. Hence, the extra internal transitions and area overhead are more than compensated for. When considering the performance impact on the external memory itself, then, although the bus-invert scheme will lead to some increase in area, due to the added invert bits, however this will not lead to an increase in power. This is due to significantly less switching activity within the memory due to the application of the bus-invert scheme.

5 Conclusions

The authors have presented a bus encoding architecture targeting a low-power SoC platform. The architecture is able to cope with the complex AHB protocol within the AMBA standard. Power analysis has been performed using 8, 16 and 32-bit bus transfers and comparisons have been made in each case to the case where bus-invert coding is disabled. Our analysis revealed significant reduction in power at the expense of a small increase in area. In

addition, the authors have analysed the switching activity trade-off when comparing the amount of reduction on data bus wires to additional activity on the invert wires.

6 Acknowledgment

The authors would like to thank Motorola, East Kilbride, Scotland for supporting this project. Dr. A.T. Erdogan would like to acknowledge the support of the EPSRC under grant number GR/NO8332.

7 References

- STAN, M.R., and BURLESON, W.P.: 'Bus-invert coding for low-power I/O', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1995, 3, (1), pp. 49–58
- MEHTA, H., OWENS, M., and IRWIN, M.J.: 'Some issues in gray code addressing', Sixth Great Lakes Symposium on VLSI, 1996, pp. 178–181
- BENINI, L., MICHELI, G.D., MACII, E., SCIUTO, D., and SILVANO, C.: 'Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems'. Seventh Great Lakes Symposium on VLSI, 1997, pp. 7–82
- FORNACIARI, W., POLENTARUTTI, M., SCIUTO, D., and SILVANO, C.: 'Power optimization of system-level address busses based on software profiling'. Eighth International Workshop on Hardware/Software codesign, 2000, pp. 29–33
- CHENG, W.C., and PEDRAM, M.: 'Power-optimal encoding for DRAM address bus'. International Symposium on low power electronics and design, 2000, pp. 250–252
- STAN, M.R., and BURLESON, W.P.: 'Low-power encoding for global communications in CMOS VLSI', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1997, 5, (4), pp. 444–455
- SHIN, Y., CHOI, K., and CHANG, Y.-H.: 'Narrow bus encoding for low-power DSP systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2001, 9, (5), pp. 656–660
- KOMATSU, S., IKEDA, M., and ASADA, K.: 'Low power chip interface based on bus data encoding with adaptive codebook method'. Ninth Great Lakes Symposium on VLSI, 1999, pp. 368–371
- BENINI, L., MACII, A., MACII, E., PONCINO, M., and SCARSI, R.: 'Architectures and synthesis algorithms for power-efficient bus interfaces', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2000, 19, (9), pp. 969–980
- BENINI, L., MICHELI, G.D., MACII, E., PONCINO, M., and QUER, S.: 'Power optimization of core-based systems by address bus encoding', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1998, 6, (4), pp. 554–562
- SHIN, Y., CHAE, S., and CHOI, K.: 'Partial bus-invert coding for power optimization of system level bus'. International Symposium on low power electronics and design, 1998, pp. 127–129
- MUSSOLL, E., LANG, T., and CORTADELLA, J.: 'Working-zone encoding for reducing the energy in microprocessor address busses', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, December 1998, 6, (4), pp. 568–572
- RAMPRASAD, S., SHANBHAG, N.R., and HAJI, I.N.: 'A coding framework for low-power address and data busses', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1999, 7, (2), pp. 212–221
- HELLMICH, H.H., ERDOGAN, A.T., and ARSLAN, T.: 'Re-usable low power DSP IP embedded in an ARM based SoC architecture'. IEE Colloquium on intellectual property, Edinburgh, UK July 2000 pp. 10/1–10/5
- PEDRAM, M., and CHENG, W.-C.: 'Memory bus encoding for low power: A tutorial'. International Symposium on quality electronic design, 2001, pp. 199–204
- STAN, M.R., and BURLESON, W.P.: 'Limited-weight codes for low power I/O'. International Workshop on low power design, Napa, California, April 1994 pp. 209–214
- BASSAM, H., and BESSLICH, P.W.: 'Ad oculos digital image processing', Student Version 2.0, Chap. 3, pp. 55–65