Hybrid Product Term and LUT Based Architectures Using Embedded Memory Blocks

Frank Heile
Altera Corporation
101 Innovation Drive
San Jose, CA 95134
frank@altera.com

Andrew Leaver
Altera Corporation
101 Innovation Drive
San Jose, CA 95134
aleaver@altera.com

ABSTRACT

The Embedded System Block (ESB) of the APEX20K programmable logic device family from Altera Corporation includes the capability of implementing product term macrocells in addition to flexibly configurable ROM and dual port RAM. In product term mode, each ESB has 16 macrocells built out of 32 product terms with 32 literal inputs. The ability to reconfigure memory blocks in this way represents a new and innovative use of resources in a programmable logic device, requiring creative solutions in both the hardware and software domains. The architecture and features of this Embedded System Block are described.

Keywords

Product terms, RAM, heterogeneous architecture.

1. INTRODUCTION

The Embedded System Block (ESB) of the APEX20K family of programmable logic devices from Altera Corporation has evolved from the FLEX10K and FLEX10KE Embedded Array Blocks (EAB) [1]. The Embedded Array Block of the FLEX10K is a highly configurable block of RAM that can also be initialized at configuration time to be a ROM block. The block contains 2048 bits that can be configured into RAMs or ROMs of the following sizes: 2048 x 1, 1024 x 2, 512 x 4 and 256 x 8. In the FLEX10KE family the EAB has been changed to 4096 bits of dual port RAM with the following configurations: 2048 x 2, 1024 x 4, 512 x 8 and 256 x 16.

In the APEX20K family of programmable logic devices, the EAB is now called the Embedded System Block because of the additional functionality of product term mode. As a ROM/RAM block, the ESB consists of 2048 bits of dual port memory that can be configured as 2048 x 1, 1024 x 2, 512 x 4, 256 x 8 and 128 x 16. For an overview of the APEX20K device family, see the Altera web-site [2] or the APEX20K data-sheet [3].

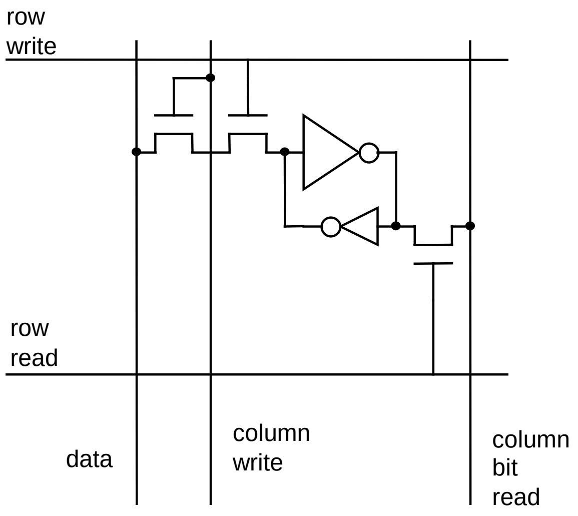
In product term mode the APEX20K ESB [4] can be configured to implement a macrocell similar to the macrocells in the MAX7000 family of CPLDs. A MAX7000 macrocell uses product terms that can be ORed or XORed together to implement logic functions. In particular, the APEX20K ESB can be configured to implement up to 16 product term macrocells with 2 product terms each at one extreme, or a 1 product term macrocell with 32 product terms at the other extreme. The level of flexibility allowed is that anywhere between 1 and 16 output product term macrocells can be implemented as long as the total number of product terms does not exceed 32. There are 32 literal inputs available in "true" and "complement" form for all of the product terms in an ESB.

An EAB in ROM mode can be used as a large many-input LUT. Wilton [5][6] and Cong and Xu [7] have provided algorithms for mapping logic into ROMs to more efficiently utilize memory blocks. They found great success in the ability of FLEX10K EABs to implement logic as ROMs. These EABs as ROMs effectively increase the density of programmable logic devices by allowing large numbers of LUTs to be combined into one EAB. This can also increase the speed of designs as large combinatorial functions with many inputs can be implemented in one level of logic in an EAB ROM, instead of requiring multiple levels of 4-input LUTs. For example, one FLEX10K EAB can implement a single output function of up to 11 inputs, or an 8-output function of up to 8 inputs. An ESB in product term mode greatly enhances this capability since, for example, the APEX20K ESB can implement a 16-output function of up to 32 inputs. The one advantage of a ROM is that it is guaranteed that it can always implement a logic function as long as the function does not exceed the ROM's input and output capabilities. An ESB in product term mode will not, for example, be able to implement all functions of 32 inputs. However, in practice, many logic functions of interest to logic designers can be implemented efficiently in product terms [8]. One confirmation of this statement is the wide popularity of product term based programmable logic devices, such as the previously mentioned MAX7000 family.

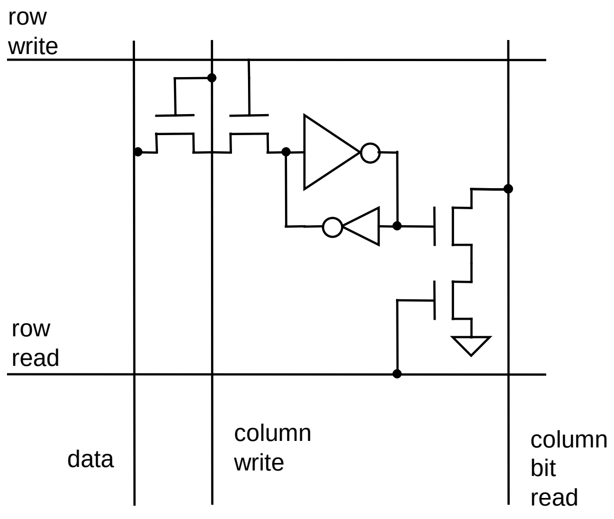
Previous work by Kaviani and Brown [9] addressed the issue of combining fixed proportions of product terms and LUT resources on the same device, and showed that the ability to split logic between these two resources resulted in a more efficient overall utilization of the device. All currently offered commercial programmable logic devices require the user to choose between an entirely product term based architecture such as Altera's MAX7000, and LUT-based architectures such as the Altera FLEX10KE and Xilinx XC4000 families.

2. PRODUCT TERM MODE

The FLEX10K EAB has 2048 bits of RAM organized as 64 rows and 32 columns. This same organization has been carried over to the APEX20K ESB. To implement product term mode for the APEX20K ESB the RAM cell has been modified, some additional circuitry has been added to the row address decoders and logic to implement a MAX7000-style macrocell has been added to the RAM output block. The RAM cell was modified by adding a pass gate so that the column bit line will only be pulled low when the cell is programmed to contain a "1" value – the bit line will never be pulled high by a RAM cell. This change is shown in Figure 1. To make this cell modification work, the buffer at the end of the column bit line is replaced by a sense amplifier with a pull-up resistor which is used to determine if any of the RAM cells are pulling the column bit line low. Thus if no cells are pulling low, the output will be "1", but if one or more cells are pulling low, the output will be "0" - thus implementing a logical NOR, which by deMorgan's inversion is equivalent to an AND gate.



(a) Original RAM cell



(b) Product term RAM cell

Figure 1 – RAM Cells

A simplified overall block diagram for a APEX20K ESB in the RAM or ROM mode of operation is shown in Figure 2. In this figure there are 16 bits of data that can be written into the RAM ($DI_{15..0}$), 11 bits of write address ($WA_{10..0}$), 11 bits of read address ($RA_{10..0}$), a write enable signal (WE) and the data that is output from the RAM ($DO_{15..0}$). In addition to the connections shown, the read and write column decode blocks need to know the mode of the ESB to correctly enable the 32 column lines used for reading and writing data. The ESB mode will be one of 2048 x 1, 1024 x 2, 512 x 4, 256 x 8 or 128 x 16. The APEX20K ESB can also optionally have the inputs and outputs registered; for clarity, these registers and the associated clocks and other control signals are not shown in Figure 2.

When the APEX20K ESB is put into product term mode, its operation changes as shown in the simplified block diagram of Figure 3. In this mode the input to the block are the 32 product term literals labeled as $D_{31..0}$. These 32 data input signals will reuse many of the signals that are the inputs to the ESB in RAM mode. As an example, the 16 bit data input bus, the 11 bit read address bus and the low order 5 bits of the write address bus can be used as the 32 literal input signals for the ESB in product term mode. The outputs from the RAM block will be the 32 signals labeled as "column bit read" which are then the inputs to the sense amps and macrocells.

Figure 4 shows the macrocell logic for the APEX20K ESB. Since there are 16 output drivers available for the ESB block when it is in RAM mode and since there are 32 product terms available it seems sensible to create 16 macrocells with 2 product terms per macrocell. The APEX20K ESB macrocell is very similar to the MAX7000 macrocell except that there are only 2 product terms instead of the 5 product terms per macrocell in the MAX7000. Another difference is that the clock and other secondary signals for the flip-flop of the macrocell are selected from an ESB-wide set of signals, which is similar to the way that the secondary signals for flip-flops in the regular 4-LUT Logic Array Blocks (LABs) are selected. In particular the ESB has 2 clocks and 2 clears and each flip-flop can independently choose from those

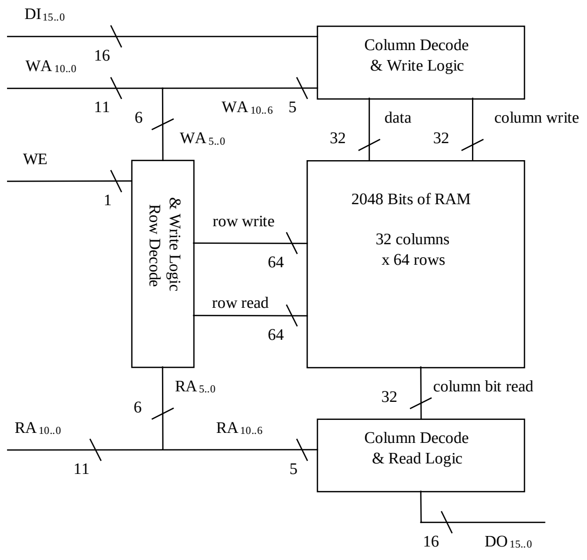


Figure 2 – ESB in RAM mode

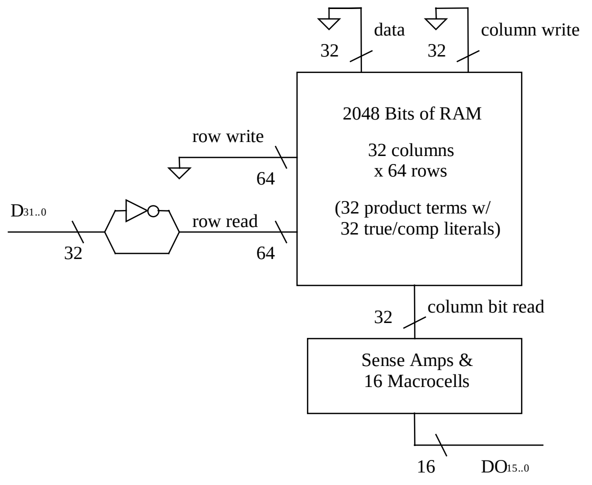


Figure 3 – ESB in Product Term Mode

clocks and clears. Of course each macrocell can also choose to bypass the register and instead output a combinatorial signal.

The product term selection matrix allows each macrocell to use anywhere from 2 to 32 product terms. When the 3rd through 32nd product terms are used, they will be borrowed from the other macrocells in the ESB and are called parallel expanders. The ESB does not implement the shared logic expander capability that is available in the MAX7000 devices. As a substitute for this shared logic expander feature, one of the product terms in each macrocell can be inverted. This allows a logic function where some number of the product terms consist of single literals to have all of these single literal product terms implemented in just one inverted product term.

In Figures 1 through 4 we have shown the major architectural features of the APEX20K ESB block. However, we have not shown many of the important details. For example, in most cases the RAM has been treated as if it was an asynchronous device. In actuality, we expect most applications will use the ESB in a synchronous manner. For example, in the RAM mode all of the read and write addresses, the data to be written and the various control signals can be synchronously registered to one or another of the two clocks available per ESB, or they can be asynchronous. We expect the synchronous mode to be used more often because

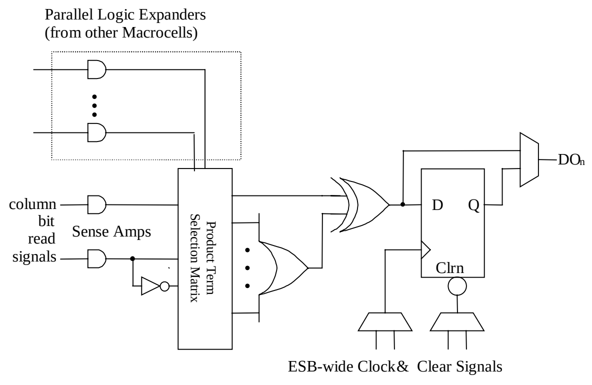


Figure 4 – ESB 2 Product Term Macrocell

the timing requirements are much easier for the synchronous design. In addition the synchronous design technique is ideal for pipelined designs.

An example of another detail we have not yet mentioned is that some muxes have been eliminated from the diagrams for clarity. These muxes will typically choose between using one or another signal depending on the ESB mode chosen. For example, if you examine Figures 2 and 3 you would see that we need a 2-to-1 mux in the vicinity of the 16 ESB outputs to choose between the RAM data read (Figure 2) and the macrocell logic (Figure 3). Another example is a 2-to-1 mux that would be needed to choose between the "row read" signals generated by the "Row Decode and Write Logic Block" (Figure 2) and the 32 literal true and complement signals (Figure 3).

3. ADVANTAGES OF ESB PRODUCT TERM MODE

The two primary motivations for implementing product term mode in embedded memory blocks are increased logic utilization and improved performance.

When we design a programmable logic device with ESBs, we want to include enough memory blocks to implement the majority of designs that use memory. However, there are always designs that use less than the full amount of memory and some designs that use no memory at all. The addition of product term mode in the ESB allows us to implement logic in what would otherwise be unused ESBs, resulting in a more efficient use of the die area. This efficiency gain is almost free, since there is very little additional circuitry required in the ESB to implement the product term mode. An alternative method for trading off memory and logic utilization is to build memories from the logic resources (LUTs) already in the device. However we find that for modern designs which have larger memory requirements the stitching required to do this imposes penalties in both the performance and utilization of routing resources, and adds additional stress on the software.

An additional benefit comes from the fact that we now have both product term and LUT-based logic on the same device. Though lookup tables predominate in the high-density programmable logic world, some forms of logic are always better suited to product term implementation than to lookup table implementation (e.g. wide-input functions and state-machines). Research by Kaviani and Brown [9] shows that significant area gains can be obtained from an architecture combining both product terms and LUTs over a purely LUT-based architecture.

On the performance side, the implementation of wide-input functions in small lookup tables causes an increase in the delay path and a corresponding performance hit. Though this problem can be alleviated somewhat with the addition of programmable cascade-style logic between the LUTs, product term implementation of this portion of the logic is a much more efficient solution for achieving the necessary delay goals.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have described a novel method for re-using memory blocks in a programmable logic device to implement product term logic. Our ideas have been implemented in the embedded system block of the APEX20K family of programmable logic devices from Altera.

With product term support we are able to better utilize die area by trading off the logic and memory requirements specific to individual designs, and can accommodate wide-input functions more suitable to product term implementation than to lookup tables. This gives both area and delay advantages over purely LUT-based programmable logic architectures. Since the bulk of the circuitry is already present in the device as embedded memory, the overhead that is required to support product term mode is minimal.

One open issue for further research is the generation of software algorithms to take advantage of these flexible resources. An interesting avenue for software research would be to study how to efficiently partition the netlist into the portions best suited for ROM, product term and LUT implementation, subject to the amount of logic and memory resources left in the part after standard memories have been generated. An additional step would be to solve the same problem subject to delay constraints, implementing the wide-input and delay-critical portions of the logic in product terms, and the remainder of the logic in LUTs. The tradeoff between product term and LUTs affects software tools from synthesis and technology mapping through place and route.

5. ACKNOWLEDGMENTS

We would like to acknowledge Mike Hutton for his valuable suggestions and help at improving this paper. However, any quality defects remaining are still the responsibility of the authors.

6. REFERENCES

[1] Altera Corporation, "Altera Data Book", 1998

[2] http://www.altera.com

[3] Altera Corporation, "APEX20K Data Sheet", 1999.

[4] F. Heile, "Programmable Logic Array Device with Random Access Memory Configurable as Product Terms", United States Patent Pending.

[5] S. Wilton, "SMAP: Heterogeneous Technology Mapping for FPGAs with Embedded Memory Arrays", Proc. ACM 6th International Symposium on FPGAs, FPGA98, Monterey, CA., Feb. 1998, pp. 171 - 178.

[6] S.Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory." Ph.D. Thesis, University of Toronto, 1997.

[7] J. Cong and S. Xu, "Technology Mapping for FPGAs with Embedded Memory Blocks", Proc. ACM 6th International Symposium on FPGAs, FPGA 98, Monterey, CA., Feb. 1998, pp. 179-188.

[8] F. Heile and A. Leaver, "Heterogeneous Technology Mapping for LUTs and Product Terms", United States Patent Pending.

[9] A. Kaviani and S.J. Brown, "Hybrid FPGA Architecture". Proceedings of the 4th International Symposium on FPGAs, FPGA 96, Feb 1996.

[10] S. Wilton, J. Rose and Z. Vranesic, "Memory-to-Memory Connection Structures in FPGAs with Embedded Memory Arrays." Proceedings of the 5th International Symposium on FPGAs, FPGA 97, Feb 1997. (Submitted to IEEE Trans. VLSI).

[11] S. Wilton, J. Rose and Z. Vranesic, " Memory/Logic Interconnect Flexibility in FPGAs with Large Embedded Memory Arrays," in CICC 96, the IEEE Custom Integrated Circuits Conf., San Diego, CA, May 1996, pp. 144-147.