Design and Implementation of a Super Scalar DLX based Microprocessor

By: Amnon Stanislavsky and Ami Schwartzman
Supervisor: Amir Nahir

VLSI Laboratory, Department of Electrical Engineering, Technion

1 Abstract

Modern SOC systems require a powerful embedded CPU as the main component of the system. Many such IP cores are available but usually at a very high cost. The goal of this project was to design, implement and fabricate a powerful microprocessor based on the DLX architecture so it can be used in future VLSI lab SOC designs as an embedded processor. The aim was to implement a CPU (the Kishon microprocessor) capable of providing high performance for a wide variety of tasks including graphic and scientific applications.

The DLX architecture, on which the Kishon is based, lacks several important features such as branch prediction, floating point and MMX instructions, efficient hazard handling, all of which degrade performance.

To overcome these limitations, the DLX architecture was extensively modified to include the following features:

- A neural network based branch predictor to reduce pipeline flushes without a significant cost in area.
- An out of order execution mechanism based on the Tomasulo's algorithm, to allow execution to proceed while processing high latency instructions
- A floating point unit with both single precision and double precision floating point support required for scientific applications
- MMX instructions that operate concurrently on packed integers for increasing performance of graphic and multimedia applications
- Support for multicore processing by enabling sharing of the data memory
- Multiple execution units to allow the execution of up to six instructions simultaneously.

The Kishon is designed to run at 100 MHz. In the final stage of the project the processor will be fabricated by Tower Semiconductors.

2 DLX Architecture

As mentioned above, the Kishon is based on the original DLX as studies in (Hennessy & Patterson, 1996).

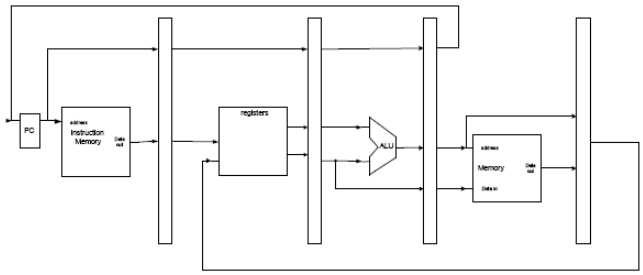

Figure 1: The DLX Architecture

The DLX is a simple yet powerful five stage pipelined RISC processor. As such it is vulnerable to control and structural hazards that occur from two commands that have data dependencies between them. The DLX's power lies in its simplicity: the classic five stage pipeline is a good solid foundation to build upon new complex features. This processor lacks basic branch prediction, causing it to lose valuable clock cycles when resolving each branch instruction. The processor can handle only integer commands making it ineffective for any application requiring floating point operations, namely graphics and scientific. Additionally, many modern multimedia applications require the use of concurrent data calculations, which it cannot handle. Finally – the single core and single datapath of the processor does not enable sufficient parallelism.

3 The Kishon Processor

3.1 Introduction

The Kishon processor was design to be a high performance embedded processor. It implements the classic Harvard architecture of separate instruction and data memories to allow simultaneous instruction fetching and data memory transactions. The Kishon was implemented as a superscalar microprocessor to allow a high degree of instruction level parallelism. The processor includes six execution units making it possible to process up to six instructions simultaneously. To reduce control hazards a novel neural based branch prediction unit was added. It provides high branch prediction accuracy with little cost in area compared to more traditional predictors. The standard integer unit uses the sixteen regular 32 bit general purpose registers. The MMX and FP units that have been added to increase performance both use the same set of sixteen 64 bit registers. The reservation stations that can be seen in Fig 2 are used for register renaming required by the Tomasulo algorithm.

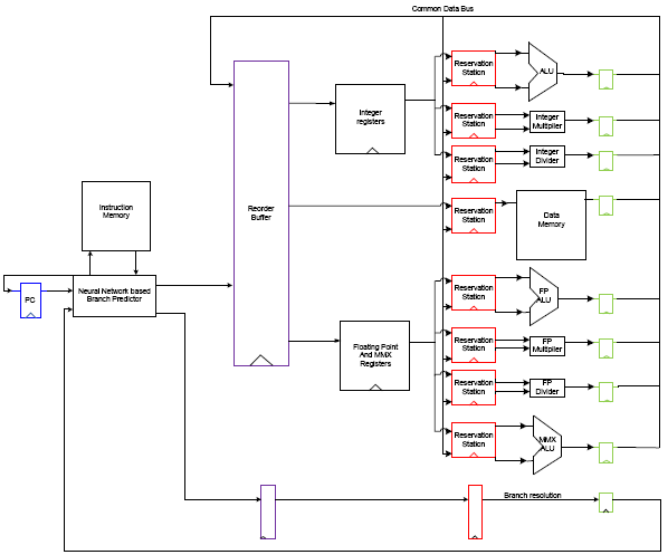

Figure 2: The Kishon Architecture

3.2 The Kishon Instruction Format

To enable fast and efficient instruction decoding, the Kishon's instruction were limited to the following formats:

I-type: integer commands that contain an immediate field
R-type: integer commands that contain operations on registers
J-type: integer program flow commands (jumps and branches)
FI-type: floating point commands that contain an immediate field
FR-type: floating point commands that contain operations on registers
MI-type: MMX commands that contain an immediate field
MR-type: MMX commands that contain operations on registers

In the following sections the main features of the Kishon processor will be described in greater detail.

4 The Kishon Main functional units and Features

4.1 Neural Network based Branch Prediction

Branch prediction enhances performance in most scenarios. The Kishon has a branch prediction unit which can be seen in the left part of figure 2 that uses neural networks to predict the branch. A neural network based branch predictor was chosen, because it proved to be more area efficient than traditional 2bit BTB as used in other processors (Jimenez & Lin).

Traditional BP units use local history to calculate the direction of the next branch. This causes to an exponential increase in memory requirements. Increased memory means more area, and longer latencies.

In the Kishon's BP unit is a *perceptron* is used to calculate the direction of the branch. In this mechanism, both local and global history is used and the memory requirements increase in a linear manner. Thus less area is used to obtain similar results and also the time needed to calculate a branch decreases. Memory requirements increase linearly with prediction accuracy.

A perceptron is used to learn correlations between particular branch outcomes in the global history and the behavior of the current branch. These correlations are represented by the weights. The larger the weight, the stronger the correlation, and the more that particular branch in the global history contributes to the prediction of the current branch.

The big performance enhancement is gained because the branch unit (effectively the entire "Fetch" stage) is designed as a zero-order branch. This means that the address of the next instruction is ready, in the cycle immediately following the cycle in which the branch is fetched. Thus in theory, a 100% success rate in calculating branch directions would enable the program to run without branches!

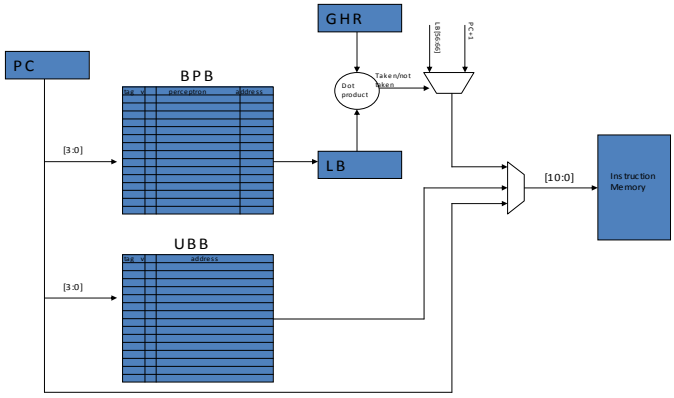

Figure 3: The Branch Predictor

4.2 Integer Unit

The integer unit performs all the common alu integer arithmetic and logic operations including integer multiplication and division.

All the sub-units were implemented using the Synopsys Module Compiler tool. The tool allows the user to specify a target clock frequency which in this case was 100Mhz. It was able to implement all alu functions in one pipeline stage except for the division operation which had to be implemented in a five-stage pipeline. As a result of this, some instructions require more than one clock cycle to perform the execution stage. It is for these instructions where the out of order execution capabilities may greatly enhance performance.

4.3 Floating Point Unit

To enhance performance of scientific applications floating point operations were implemented. The floating point unit uses the standard IEEE representation for floating point numbers:

$$(-1)^s \times 1.m \times 2^{e-127}$$

where "s" represents the sign, "m" the mantissa and "e" the exponent. 8 bit mantissa and 23 bit exponent for single precision and 11 and 52 respectively for double precision. The "round to nearest" method of rounding is used for all operations.

The FPU performs all operations for both single and double precision. The unit implements, addition, subtraction, comparision, multiplication, division, square root and integer to floating point conversion.

The design and implementation of such a floating point unit from scratch is an enormous task. Therefore it was implemented with the aid of the Synopys DesignWare libraries. DesignWare provided parallel implementation of the floating point operations. Unfortunately the units did not meet timing requirements, therefore manual pipelining was performed for each operation till the 100Mhz operating frequency was met.

Despite all efforts, the division and square root units were so large that no amount of pipelining enabled the unit to function at a 100 MHz clock frequency. The solution to the problem was to reduce the clock frequency of these two particular operations to 25Mhz. Thus the effective number of cycles is simply the number of cycles with a 25 MHz clock times four for the main 100 MHz clock.

The FPU requires special 64 bit registers. The sixteen 64 bit register file is also shared by the MMX unit described below.

4.4 MMX Unit

Many graphic and multimedia applications use 8 or 16 bit data types. A significant speedup of these applications can be achieved with the use of SIMD (single instruction multiple data) instructions (also known as MMX instructions). Fig. 5 illustrates how four 8–bit addition instructions can be performed in a single cycle simply by packing the four 8-bit operands into a 32-bit register. The MMX unit implements over 50 MMX instructions, which include addition, subtraction and multiplication of various sized packed data units ranging from 8 to 32 bits. This unit also implements the instructions that perform the packing and unpacking of the data units that is required for the different application.

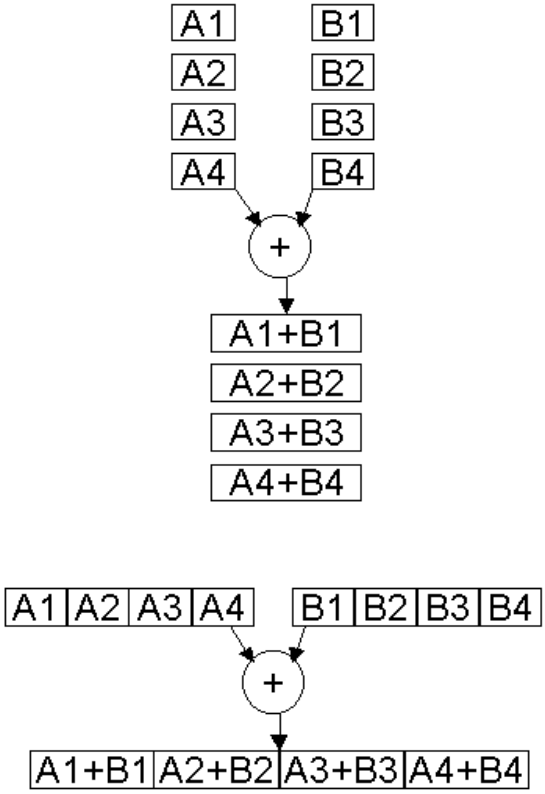

Figure 4: MMX example

Due to its complexity the MMX unit was also implemented with the aid of the module compiler. The unit met the 100Mhz clock cycle requirement without the need of pipelining any of the MMX instructions.

4.5 Out of Order Execution

Out-of-order execution, OoOE, is a paradigm used in most high-performance microprocessors to make use of cycles that would otherwise be wasted waiting for high latency instructions to complete.

In in-order processors, the processing of instructions is normally done in these steps:

1. Instruction fetch.
2. If input operands are available (in registers for instance), the instruction is dispatched to the appropriate functional unit. If one or more operands are unavailable during the current clock cycle, however, the processor stalls until they are available.
3. The instruction is executed by the appropriate functional unit.
4. The functional unit writes the results back to the register file.

The out of order paradigm based on Tomasulo's algorithm that was implemented on the Kishon, breaks up the processing of instructions into these steps:

1. Instruction fetch.
2. Instruction dispatch to a reservation stations (RS).
3. The instruction waits in the station until its input operands are available. The instruction is then allowed to leave the station before earlier, older instructions.
4. The instruction is issued to the appropriate functional unit and executed by that unit.
5. The results are queued in a reorder buffer (ROB).
6. Only after all older instructions have their results written back to the register file, then this result is written back to the register file through the common data bus (CDB).

The ROB is responsible for decoding and dispatching the instructions and for in-order completion. In-order completion is important because the BP unit may mispredict branches at times in which case, all the changes that have occurred need to be undone. The ROB also implements a register renaming mechanism which removes several data hazards.

The CDB connects the ROB to the RSs. The RSs write the result of their functional unit to the CDB so both other RSs and the ROB can read the result. The ROB monitors the CDB for completed instructions and updates its registers. The RSs monitor the CDB for operands needed for an instruction. In this way, the result may be accessed before it is written to the register file (commonly known as forwarding or bypassing) thus saving additional clock cycles.

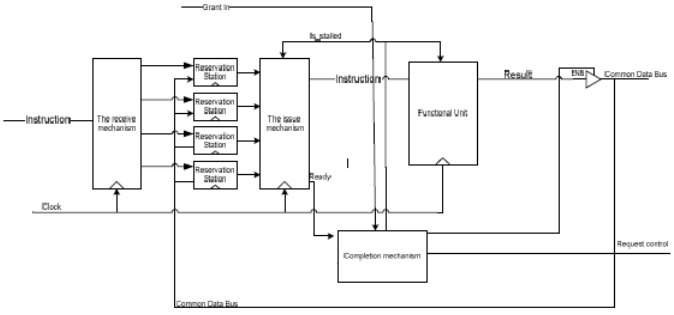

Figure 5: Kishon's ROB datapath

4.6 Multicore support

Multicore processing capability can further boost performance and can be achieved by allowing the different cores to access the same shared memory. The Kishon supports multiple cores using the MISD (Multiple Instruction streams and Single Data stream) model.

According to this model, each core has its own instruction memory. Each core can be loaded with different instructions and all cores can share the same data memory.

Each core can perform an instruction fetch on every clock cycle but can access the data bus only after being granted permission by the memory controller.

This memory controller also serves as a connection to the "outside world" via the chip's IO pads.

5 The Kishon's Instruction Flow

As previously stated, the Kishon implements a four stage pipeline. All instructions are typically executed in the following four stages.

1. Fetch:

The PC (program counter) provides an address. The branch predictor looks it up in its internal memories. If found, it uses the information to determine the address of the instruction following the branch. If not found, the address from the PC goes directly into the instruction memory. The command is fetched from the instruction memory and is transferred to the Reorder Buffer.

2. Decode & Dispatch:

If the ROB (reorder buffer) and the associated reservation station do not have a free slot the instruction is stalled. Otherwise, the reorder buffer allocates the command an address and decodes it. After the required registers are fetched from the register file, the command is written to a reservation station which is part of a reservation station cluster for a specific functional unit. If the operands are not available, the reservation station monitors the CDB for the required operand. The reorder buffer (purple register[1]) monitors the CDB (common data bus) and updates the required registers accordingly.

3. Execute:

A reservation station issues a command that has all its operands available to the operational unit. When a command completes it writes its result to a buffer and requests an arbiter for access to the CDB (common data bus). If access is not granted, the entire functional unit is stalled. Otherwise the unit is freed and its reservation station frees the slot previously used.

4. Commit:

When a functional unit receives access to the CDB (common data bus), the result from the buffer is written to the CDB and the ROB reads it's data and writes it to the correct address of the destination register. When the ROB has in-order completion of instructions, it updates the register file with the values

6 Project Challenges and Achievements

The main task of the project was to design and implement a highly complex microprocessor in a short period of time. As previously explained, numerous complex units were designed and integrated into the system to provide good performance for a wide range of applications. It would have been impossible to complete the processor in the allocated time frame had all the units been designing manually.

After evaluating the available options it was decided to use a tool called Module Compile to implement the MMX and integer units. Module Compiler is a program provided by Synopsys and it is a CAD tool specifically designed for datapath construction. It is capable of pipelining a design so as to meet timing specifications. This program uses a proprietary HDL language called MCL (Module Compiler Language) which is very similar to Verilog. A mechanism called "Directives" is used to inform the tool which attributes are important. In this case – the timing specification had to be met at the cost of area and power consumption.

Another very useful aid to this project was DesignWare. DesignWare is a library of IP cores available to the user. Among those IP are logic units, arithmetic units, floating point sub-units and many more. It was decided that the best and fastest way to build the floating point unit was with the aid several DesignWare IPs. These resulting units were then manually pipelined to meet timing constraints. The memory controller and reservation station cluster also contain a DesignWare IP, namely the arbiter.

The design of the register files proved to be another very challenging issue. Normally register files are implemented using standard registers. The Kishon has two large register files. Had they both been implemented using standard registers, the area cost would have been very high. It was decided to implement them using dual port SRAMS. The problem with this implementation was that the register files required three ports, two for reading and one for writing. The solution chosen was to use two identical units. Data is written to both units (so the information in both is identical) using one port of each unit. The two free ports (one from each unit) are used to read the two operands. Despite the redundancy, this implementation consumes less area than one using registers.

Additionally, the novel neural network based predictor that provides high prediction accuracy at low area costs, out of order execution for stall reduction, superscalability for high instruction level parallelism, mutilcore capability enabled by a versatile memory controller are all advanced features that make the Kishon a very capable processor.

7 Simulations

Simulations show that the processor is fully functional. Test programs were created and run to verify correctness of the various features, namely branch prediction, ooo execution and FP and MMX operations. The post synthesis timing verification predicts that the silicon should meet timing requirements of 100Mhz operating frequency.

The Kishon was also synthesized and implemented on an Altera Cyclone II FPGA card to verify and to demonstrate its functionality. The chip is due to be submitted to Tower Semiconductors for prototype fabrication.

8 Synthesis and layout

The chip was synthesized with the Synopsys Design Vision tool using the Tower Semiconductors 0.18 micron 6 layer metal CMOS technology. The SRAM memories were implemented using the Tower DP SRAM generation program.

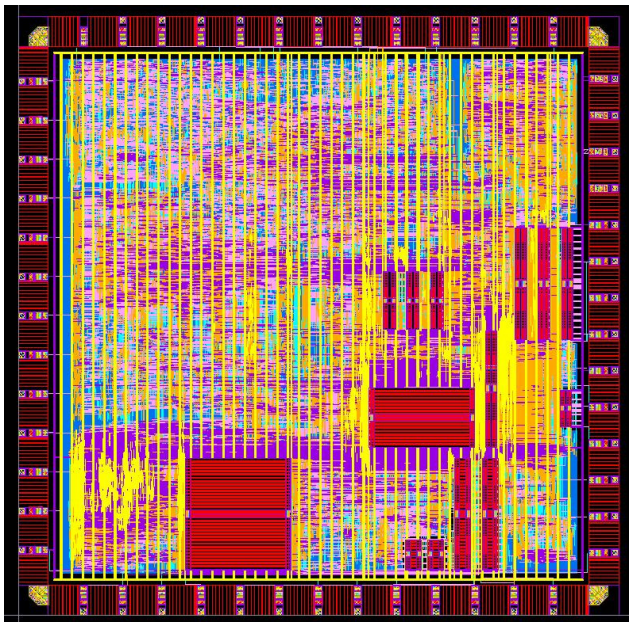

Figure 6: The Kishon's Layout

The layout (see Fig. 6) was implemented using Cadence's SoC Encounter layout tool. It was then transferred to the Virtuoso program for layout verification test such as LVS and DRC.

The striped blocks are on-chip memories. The largest is the instruction memory and next to it is the data memory. The other memories that can be seen are the two register files that are implemented using three memories each and the memories that the branch predictor and reorder buffer use. The dimensions of a single core are 5x5 squared mm.

9 Conclusions

This ambitious project to provide a complete high performance embedded microprocessor for the VLSI lab proved to be very large and complex that would normally require a much larger team to implement. Only deep knowledge and efficient exploitation of sophisticated CAD tools and methodologies enabled the small design team (2 students) to complete the design in the allocated time frame.

In performing this project we accumulated vast knowledge and experience in the field of VLSI chip design in all stages of a complete design flow ranging from the concept itself right up to the final stages of tape-out and fabrication.

10 Acknowledgements

We would like to thank Goel Samuel for all of his time and support. We would also like to thank our supervisor Amir Nahir for his guidance throughout the project and for the verification part in particular.

11 References

Hennessy, & Patterson. (1996). *Computer Architechture, a quantitive approach.*

Jimenez, D. A., & Lin, C. (n.d.). Dynamic Branch Prediction with Perceptrons.

RISC. (2007, February). Retrieved 2008, from WikiPedia: http://en.wikipedia.org/wiki/Risc