# The Design and Implementation of an Asynchronous RISC Microprocessor

By Raanan Zacher and Raz Kivelevich-Carmi
Supervisor: Dr. Rakefet Kol

VLSI Laboratory.
Department of Electrical Engineering
TECHNION - IIT

## Abstract

In a wide range of computer controlled systems the issue of security is often of vital importance. System developers invest a vast amount of effort in trying to create a secure system while hackers invest a great deal in trying to breach this security. Many methods of violating security have been devised. Recently, a clever method, based on inserting spurious signals into the clock line of a microprocessor that enabled secure data to be extracted from the system, was discovered. Several different approaches could be taken to tackle this problem. After some consideration it was decided that eliminating the clock from the system could be a very successful solution to the problem. The aim of this project was to design, simulate, implement and test an 8-bit asynchronous RISC microprocessor. The chip was designed over a period of 3 months and was submitted for fabrication at CMP in France. It was designed using a 3 metal 0.6u CMOS process, the die was pad limited with a size of 3.6 X 3.6 sq. mm. Timing simulations show that the chip should have a maximum throughput equivalent to about 100MIPS.

The circuit was designed and implemented in the VLSI laboratory at the TECHNION, using the Synopsys and Cadence design tools and prototypes were fabricated through CMP in France.

## 1. Introduction

The large majority of all designs are implemented as synchronous designs. Synchronous systems have many advantages. Many CAD tools have been developed based on the assumption that the design is synchronous thus allowing engineers to design and implement very complex systems that could not have been implemented otherwise. However, with the advance of technology, problems that were once considered minor, have become major obstacles in synchronous designs:

1. Clock skews – in a large complex design, the clock might reach different points of the circuit at different times because of the increasing delay that results from the relative increase of wiring length.
2. Clock distribution – distributing the clock throughout the system consumes a substantial amount of silicon area.
3. Signal skews – at high frequencies, signal skews become longer than the clock cycle.
4. Power consumption – in modern designs, the clock rate is very high. Toggling of the clock signal of a flip flop causes power consumption even if the value of the flip flop remains unchanged.

The alternative to synchronous design is asynchronous design. This design approach has been around for many years but is not very popular with designers because of the following challenges:

1. Higher design complexity.
2. A lack of suitable commercial CAD tools.
3. Usually requires more hardware to implement.

Despite the difficulties there are cases where asynchronous design can yield better results that respective synchronous designs. Asynchronous circuits have no clocks so all problems related to the clock, like clock skew and clock distribution are eliminated, power consumption is usually lower and computations are usually performed at average case speeds and not worst case speeds.

## 2. Motivation

Choosing to implement an asynchronous microprocessor did not arise from any of the traditional considerations stated above, but due to an entirely different reason. Recently, it was discovered that the security of a particular system could be compromised, simply by manipulating the clock signal of the controller of the system. By inserting spurious signals into the clock line of the controller, secure data could be extracted from the system.

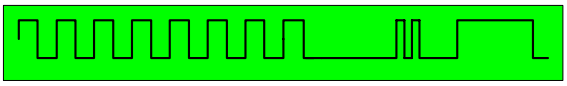

Fig 1: Spurious signal inserted into clock pin

This manipulation of the clock signal also enables the hacker to skip over the execution of instructions vital to the security of the system.

Several methods of overcoming the security violation were considered. One option is to generate a clock signal internally within the chip and use this signal to clock the microprocessor. The problem with this solution is that, in the available technology, the clock frequency of an on chip oscillator varies considerably and may cause the chip to malfunction. Additionally, it would be very difficult to synchronize external data/instructions with the clock. Another option could be to try to identify all types of security violations possible using the clock line and try to devise a method of preventing each one. Dummy clock signals could be added to make security beaches more difficult. None of these ideas appear to give a comprehensive solution to the problem. Finally, it was decided that a promising approach to take was to replace the microprocessor of the system with an asynchronous one. Clearly, this totally eliminates any possibility of manipulating the clock signal.

## 3. Challenges

Designing an asynchronous processor proposes many challenges. Most of the challenges are associated with the fact that there are few academic tools and no suitable commercial tools readily available to the asynchronous designer. Therefore, there is no way other than to adapt the use of existing traditional synchronous design tools in order to allow for them to be used for asynchronous design.

Another challenge is related to the low number of asynchronous processors implemented that could be used as references. To date there are less than a dozen such processors only one of which is commercial.

The asynchronous approach is different from the commonly used synchronous design. It is a great challenge to compare the different approaches and to choose a methodology to successfully implement an asynchronous architecture.

In order to confront these challenges a method in which commercial tools can be used was needed. The solution proposed was a hybrid approach of using high level HDL design and gate level schematic editing. HDL tools were used for the design of complex modules. The modules were then synthesized and a schematic editor was used in order to modify the synthesized circuit to fit the design requirements.

The design of the asynchronous processor required a different line of thought. As the entire design was not based on a standard methodology, the process of designing the chip, involved a continuous modification of the architecture until it fulfilled both the requirements and the constraints.

## 4. Fundamentals of Asynchronous Design

The common feature of all asynchronous systems is that they do not have a clock signal that synchronizes all operations performed by the system. A typical asynchronous system is composed of units that communicate between themselves using a handshake protocol (see Fig 2).

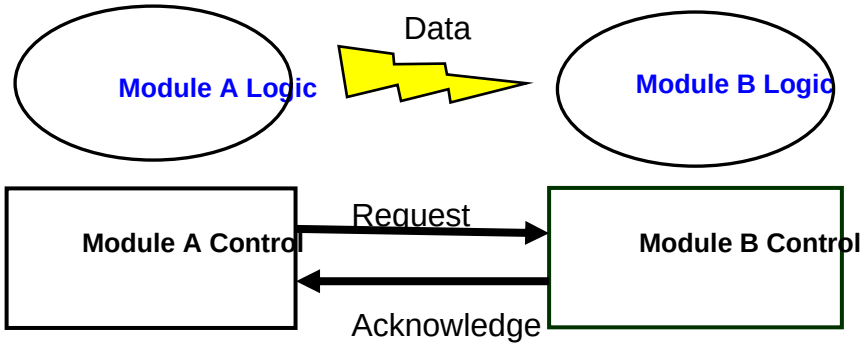


Fig 2: Typical asynchronous system composed of two modules

The handshake protocol enables module A and B to communicate. Module A notifies B that the data is ready and module B acknowledges that it has received the data. This protocol can be implemented in four (see Fig 3) or two phases (see Fig 4).

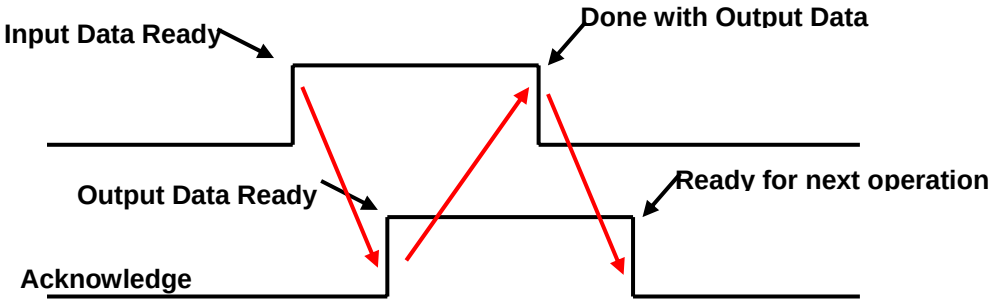


Fig 3: Four-phase handshake protocol

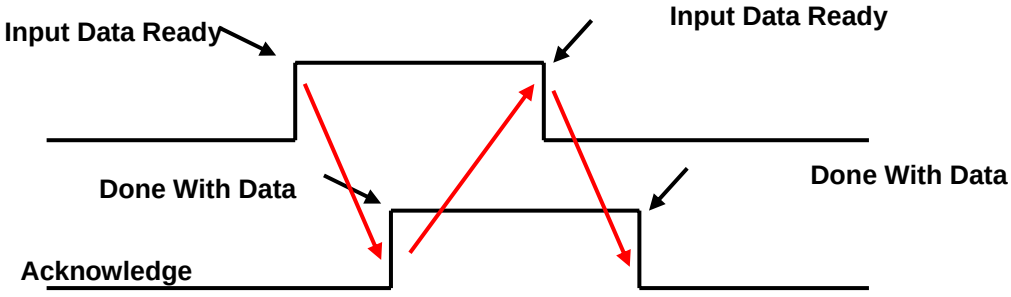


Fig 4: Two-phase handshake protocol

Several design methodologies asynchronous systems have been developed by several researchers over the years including self-timed, bounded or matched delays, handshake circuits, dual or single rail all of which can be combined with the two or four phase handshake protocol. Describing all these methods is beyond the scope of this paper. After evaluating all these methods and considering the tools available, it was decided to implement the design using matched delay, four phase single rail technique.

## 5. Specifications and Requirements:

The designed processor is complex system capable of executing a set of commands, stored in an external memory device. The processor accesses the program memory using a four phase, single rail protocol. The external memory is a standard memory device, modified, in order to implement the

asynchronous protocol (the modification is an external addition of matched delay and handshake lines to the memory device).

The processor should allow communications with other external devices via four I/O ports, two of which are to be used for data transfer while the other two for conducting an asynchronous four phase single rail protocol.

The processor consists of several major internal resources, which enable it to execute the fetched instructions. Table 1 summaries these internal resources:

| Resource | Description |
|---|---|
| Port A (PA) | 8bit Port A register |
| Port B (PB) | 8bit Port B register |
| Port C (PC) | 8bit Port C register |
| Port D (PD) | 8bit Port D register |
| RAM 16x8 | Internal memory of 16X8 bits |
| Status Register(PSW) | Processor's flags register |
| Accumulator (ACC) | 8bit wide accumulator |
| ALU | Arithmetic Logic Unit |

**Table 1 – Internal Resources**

Being a RISC microprocessor, the instruction set is very limited. It consists of 8 op-codes and is a one-address machine. Each instruction is two bytes long, one for the op-code and the other for the operand. The instruction set can be divided into three main types:

- Move instructions
  - Move data from memory to ACC
  - Move data from ACC to memory
  - Move immediate data directly to ACC (Operand to ACC)
- Jump instruction
- ALU instructions

## 6. Architecture and Implementation

The processor consists of several major components, each with its own purpose and each capable of negotiating data transfers with its neighboring units. The negotiation is done via a four phase asynchronous protocol. Since the data flow through the system is governed by the system's units, there is no need for a central control unit (like the ones found in most synchronous processors), instead, the control is distributed throughout the chip. The only similarity to the traditional control unit, is the part responsible for initiating instruction execution. From this point on, the instruction flows independently through the units. When the execution is completed, the control unit fetches the next instruction to be executed.

The processor architecture can be divided in to three major blocks (see Fig 5) – the ALU, the internal memory space and the flow control subsystem, which consists of the initiating control unit and the program counter.

**The ALU**

The arithmetic logic unit and its peripheral sub-units (ACC and PSW) are the functional heart of the system. The ALU is involved in most op-code executions.

**ALU sub-units:**

In addition to the ALU logic, the ALU contains two additional blocks, the ACC (Accumulator Register) to store the results of a computation and the PSW (Status Register) to store the present status of the processor.

**The Internal Memory Space:**

One of the processor's requirements is to map each of the internal resources on to the memory address space. As a result of this requirement, the internal memory space contains both the actual internal data RAM (16x8 bit) of the processor and the I/O ports.

In addition both the ACC and PSW (which are part of the ALU block) are mapped on to addresses in the internal memory space, allowing the programmer to access them directly as memory cells.

**Flow Control Subsystem:**

The purpose of the flow control subsystem is to fetch instructions, decode them and initiate their execution. This unit contains the PC (program counter), the Instruction Register and a small control unit.

**The PC:**

The PC's function is to put the next instruction's address on the external memory device address port. The PC also contains the mechanism required to perform the jump instruction (one of the instructions which are not executed by the ALU) and for incrementing the address.

**The Instruction Register:**

Consists of two registers – one for storing the op-code and the second for storing the operand.

**The control unit:**

The control unit decodes the fetched op-code and initiates the flow of instruction execution, by sending control signals to first units in the execution chain.

As described earlier, one of the major challenges in designing an asynchronous system is the lack of commercial tools. In order to overcome this problem, there was a need to invent new ways of using the traditional synchronous design tools, and to adapt them to asynchronous design.

One adaptation was to initially use standard high level HDL techniques to design, simulate and synthesize the system. After synthesis, the design was then manually modified in order to incorporate all changes required for asynchronous implementation. The modified system was then simulated again at the gate level. Fig. 5 illustrates the ARP architecture.

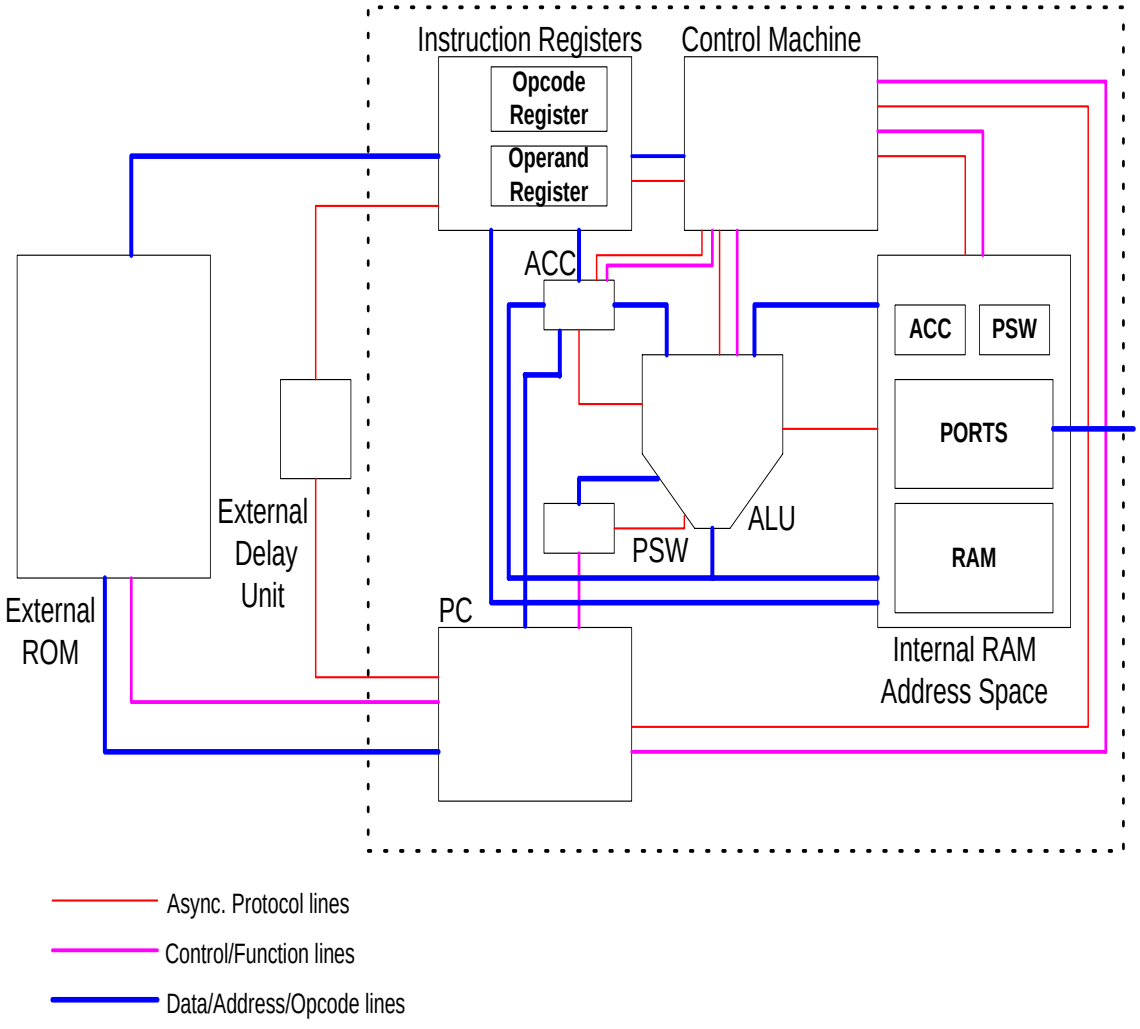


Fig 5 : The ARP architecture

## 8. Instruction Execution Flow

As partly described in previous sections, a command flows through the system as follows:
1.a. The PC issues an address to the program memory.
1.b. The PC triggers the latch of the op-code and operand through the instruction register.
2. The instruction register stores the op-code and the following operand.
3. The instruction register triggers the control unit.
4. The control unit decodes the op-code (stored in the op-code register inside the instruction register).
5. In case of an instruction involving memory, the control starts the negotiation with the memory in order to latch an address (the address is the operand data, stored in the instruction register).
6. After receiving a completion signal from the memory, the control unit begins negotiation with the ALU (or internal ACC). In addition an execution command to the ALU is issued.
7. The ALU, performs the functional part, in accordance with the decoded op-code, provided by the control unit. The ALU negotiates with the RAM, ACC and/or PSW (depending on the command being executed).
8. The ALU sends a completion signal to the control unit (ending the command execution flow).
9. The control unit issues a "begin next instruction" to the PC.
10. The PC increments the address the process is repeated.

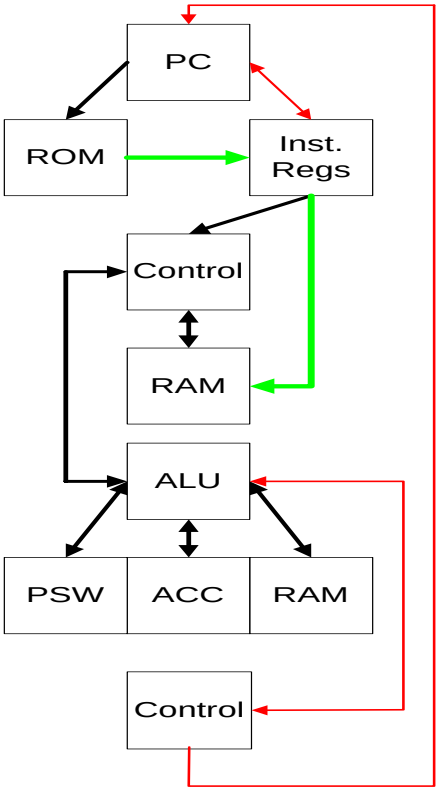


Fig 6 : The Instruction Flow

## 9. Simulations

Initially, before any detailed design was done, a block-level simulation of the system was performed to check the correctness of the asynchronous interfaces between the main modules. During the design, each module was simulated separately to ensure that it functioned correctly before being integrated with other modules. When the design was fully integrated the chip was systematically simulated to ensure that its functionality was fully simulated. All commands were checked and a complete test program was run successfully. Timing verification was run to verify timing and to have an accurate estimate of the expected speed of the design. It showed that the chip should have a maximum throughput equivalent to about 100 MIPS.

## 10. Layout and Layout Verification

The layout was generated using Cell Ensemble and Block Ensemble of the Cadence Design tools. These are automatic place and route tools that were used to create the floorplan, place the I/Os and standard cells and perform the detail route of the cells. Design rule checks and layout versus schematics checks were run to verify that the layout had no errors. These checks revealed some errors that had to be manually corrected. The corrected layout was submitted for fabrication. The final size of the chip was 3.6mm X 3.6mm and was in fact pad limited as can be seen in Fig. 7.

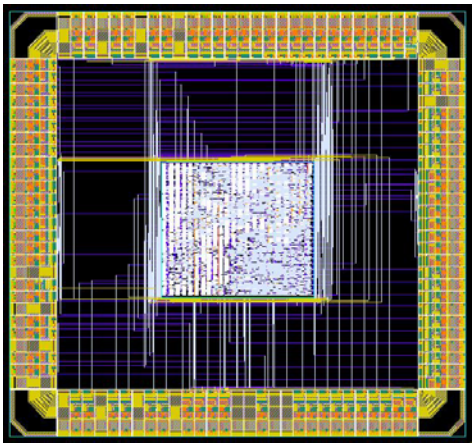

Fig 7 : The final layout

## 11. Conclusion

A fully customized asynchronous microprocessor was designed and implemented in a relatively short period of time. The chip meets all specifications. It will help in preventing serious security violations that would have been difficult to prevent by other methods.

## References

[1] Alexander Yakovlev, Designing Self-Timed Systems, VLSI Systems Design, September 1985.
[2] Charles L. Seitz., System timing, Chap. 7, pg 218-262, Introduction to VLSI, Mead and Conway.
[3] Ward, S.A., Halstead, R.H., Jr. Computation Structures.
[4] Tutorial slides from the Async 2000 Conference, Apr 2-6, Eilat, Israel.

## Acknowledgments

We would like to thank Rakefet Kol who supervised the project and Goel Samuel for his extensive support.