Speed Increasing by Structural Optimalization, at RISC Processors

Péter Pálfy [1], Timót Hidvégi [2]
Széchenyi István University, Egyetem tér 1, H-9026 Győr, Hungary, www.sze.hu
[1] iplan@freemail.hu
[2] hidvegi@sze.hu

1 Introduction

Increasing speed of data processing at microprocessor structures is a very complex question. At most of the developments, the main directive is to speed up our systems by the acceleration of architectural elements as individuals, at the deepest elementary levels, which is possible. My main interests are improving systems by only system level optimalizations, without speeding up our elements. In this work I would like to show some alternative solutions, which can increase speed at a microprocessor[1], [3], in our case at a microcontroller.
The slogan of this topic could be: „ Parallelization at Different Levels”.

1.1 The Ways of Alternatives (Alternative Speeding up Techniques)

- Transforming the internal bus system, to increase the „work content of one instruction”.
- Give self intelligence to single elements
- Equalizing system delays, by low level system optimalizations, beside speeding up at
- different elements at the lowest level.
- As an issue of the previous points, one cycle instruction performing – no pipeline required
- Instruction-controlling by direct controller bits – Wide program words.
- Speeding up by multiplicating elements, not the whole processor.
- Element multiplications, optimalized work distributing between elements.

Some of these solutions can be used by itself, or combinated at different ways. In the followings some elements of an optimalized combination will be seen, as a simple microcontroller structure.

2 Our Experimental Device

We perform experiments about implementing some alternative methods. In this work our device is a FPGA [4], [5] – a special configurable / programmable hardware device – it is for example a Spartan-II E. It contains Complex Logic Blocks, which are programmable logical circuits, some RAM-memory, and configurable I/O ports.

2.1 Characteristics of our Device (XC2S800)

Number of logic cells:	15552
CLB array:	48 x 72
Maximum available I/O-s:	514
Available system gates:	600,000
Block Ram:	288kBit

2.2 Internal Structure of Spartan FPGA

2.2.1 Location of the Architectural Elements in the Device

This Figure 1 shows the locations of the internal logic blocks (CLB-s) [6], [7], [8].

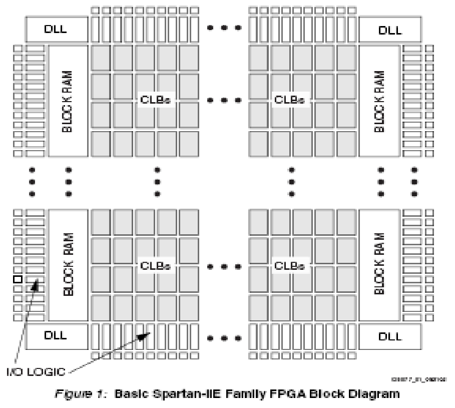

Figure 1
Basic Spartan-II FPGA

2.2.2 Connecting the CLB-s

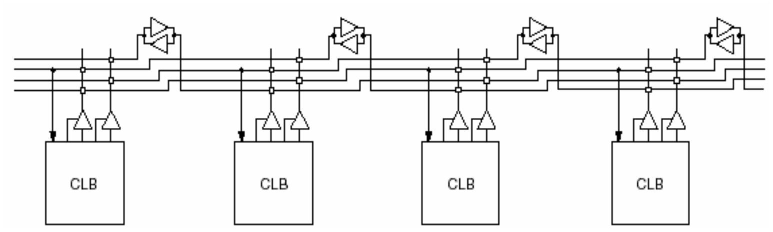

Figure 2
Buffer Connections between the CLBs

The Figure 2 is about the connections of the CLB-s. They are connected to each other by tri state buffers. The assembler and optimalizator software determines these connections, and configure these cells, based on a source program.

2.2.3 CLB from Inside

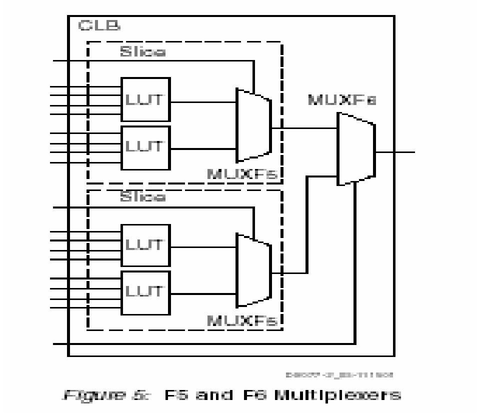

Figure 3
Logical Units in CLBs

As it can be seen in the Figure 3, a CLB contains 2 Slices, which contain 2 LUT-s (Logical Units). The assembler can select one of four logical functions by multiplexers.

3 Bit High Speed Microcontroller Structure

3.1 An Alternative RISC Microprocessor

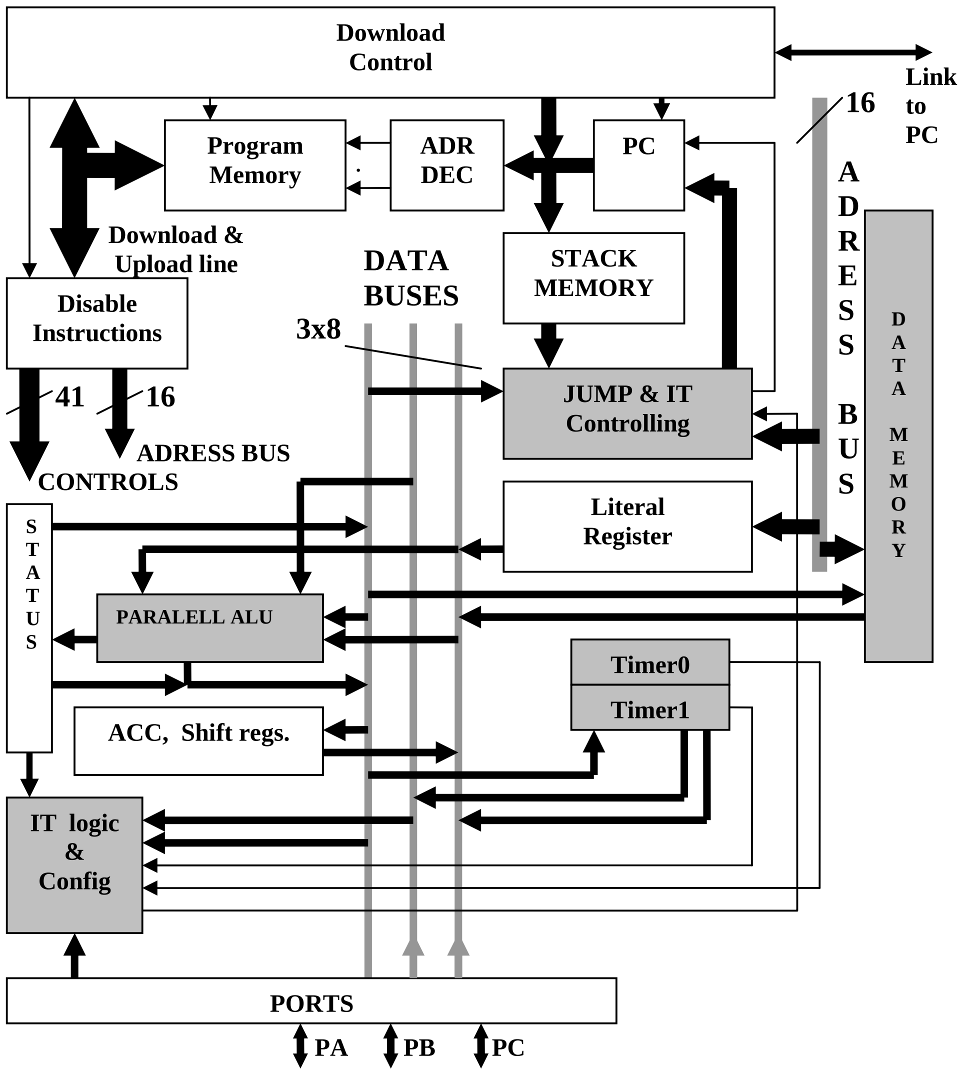

Figure 4
Architecture Block Schema

The Figure 4 shows an alternative of structural speed increasing. The grey boxes are the most important elements of reorganizing. These elements are the 3 bus data foam system, and the separated address bus. The other elements of the optimalization are the intelligent memory, the parallel ALU, and the special controller Unit. Beside these extras, the solution contains 2 programmable timers, and a programmable IT system. Each bits of port B can be used as an IT, and the both of the timers can produce a kind of „overload” IT. Status IT is available too, for example when it is necessary to assign an it for the overload in the data operation.

3.2 Speed up Solutions

3.2.1 Strengths of the 3 Bus Data Foam

The main power of this architecture is in the one cycle instruction performing.

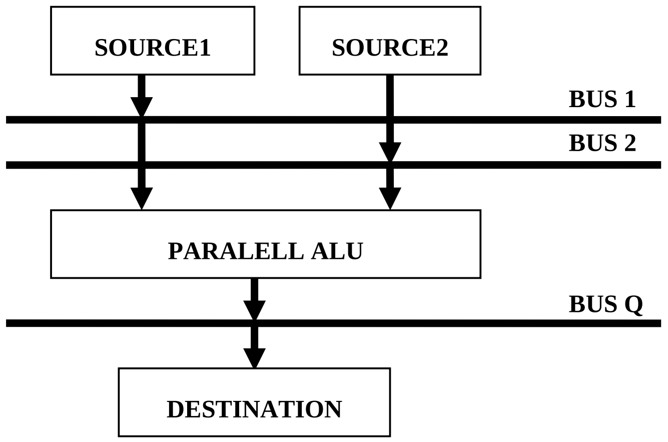

Figure 5

Performing Operations with Three Data Bus

The words, which can describe the work of this system well are data guiding. By the program words, two source, the destination device, and the operation must be selected. As an issue of this structure, the two steps of performing are associated with clock edges. These steps are reading rom source and writing into destination. The longest system delay in this controller is the ALU-delay, it determines the maximum system clock frequency. By this method, actually as fast device is available to built, as well the developer can optimalize the ALU-speed. As the result of this structure reorganizing, the system requires special instructions and special assembly, and of course a special assembler.

3.2.2 Instruction Set

3.2.2.1 The Components of a General Instruction

DATA INSTRUCTION → SOURCE1,SOURCE2,DESTINATION → #ADRESS INSTRUCTION → ADRESS

In the same instruction line, the operation can be performed, and taken into the destination device. Beside this, a memory rack can be addressed, or it is available to write an address of a program word into the appropriate address register, in the same line – because of the separated address bus. The other advantage of this system is, that the number of the move instructions is minimal, so beside the large information content of the program words, it is available to spare time by omitting to use move instructions. Of course the move instruction is usable if it is necessary.

3.2.2.2 Some Example for „How to Use

```
0029H    ADD      SR,PORTC,ACC        #RDM     20H
0030H    AND      ACC,PORTA,PORTC#LWR          10010100B
0031H    ADD      ACC,L,PORTA         #JWR     34H
0032H    JMPIF    MI #JWR                      32H
0033H    JMP                          #JWR     34H
```

This routine shows us a memory reading. If a memory read is required, an #RDM instruction must be used in the address line. In this example, the delay of the intelligent memory is less than three instruction cycles. In this case three instructions can be inserted into the source program, before the memory result is accessible. In the reading of the memory read register, the MI-bit (Memory Is ready), is at „high", when the asked data is available to read.

3.2.2.3 The DATA Instructions

Arithmetic1	ADD	S1,S2,D	Adds the two source, and store in dest.
Arithmetic2	SUB	S1,S2,D	Subtracts S2 from S1, and store in dest.
Logic1	AND	S1,S2,D	AND logic between the bits of the two operand.
Logic2	OR	S1,S2,D	OR logic between the bits of the two operand.
Logic3	XOR	S1,S2,D	XOR logic between the bits of the two operand.
Logic4	NOR	S1,S2,D	NOR logic between the bits of the two operand.
Logic5	NAND	S1,S2,D	NAND logic between the bits of the two operand.
Logic6	NXOR	S1,S2,D	NXOR logic between the bits of the two operand.

Logic7	-AND	S1,S2,D	$\overline{S_1} \bullet S_2$ logic between the bits of the two operand.	
Logic8	AND-	S1,S2,D	$S_1 \bullet \overline{S_2}$ logic between the bits of the two operand.	
Logic9	-OR	S1,S2,D	$\overline{S_1} + S_2$ logic between the bits of the two operand.	
Logic10	OR-	S1,S2,D	$S_1 + \overline{S_2}$ logic between the bits of the two operand.	
Logic11	INV	S1,D	Put the inverted of S1 into destination.	
Logic12	INV	S2,D	Put the inverted of S2 into destination.	
Logic13	MOV	S1,D	Moving S1 to destination.	
Logic14	MOV	S2,D	Moving S2 to destination.	
Logic15	FF	D	Set the all bits of destination.	
Logic16	CLR	D	Clear the all bits of dest.	
Shift1	SR	S1	Shifting S1 right, without Carry	
Shift2	SL	S2	Shifting S2 left, without Carry	
Shift3	Operation		S1, S2, SR	Shift result right with Carry
Shift4	Operation		S1, S2, SL	Shift result left with Carry

3.2.2.4 Address Bus Instructions

#MWR	address	Write into data memory
#MRD	address	Read from data memory
#JWR	address	Give a jump address
#IWR	address	Give an IT address
#LWR	konst	Write into Literal register
#NOADR		No address

3.2.2.5 System Instructions

SCLK Stop program counting, address operation is prohibited (Sleep-kind).

It's similar as the sleep operation in other controllers, but we can set a direct operation, if it is active.

In this case, the processor behaves as a simple combinational network.

3.2.3 The Configurable IT Logic

3.2.3.1 Description and Block Schema

With this logic, the bits of portB are defined as interrupt inputs, and beside this the status bits can be used as internal interrupt bits.

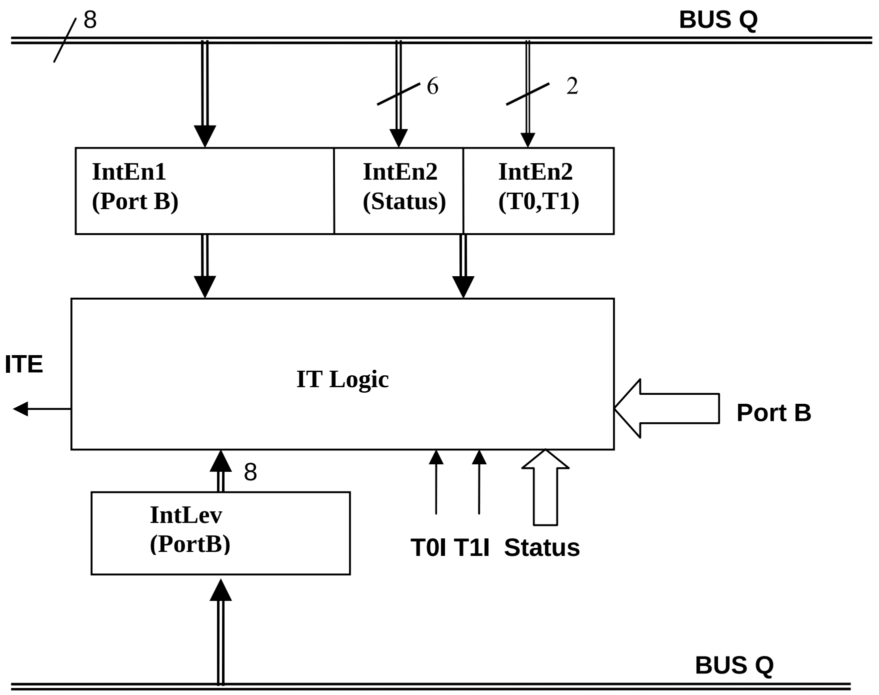

Figure 6
Interrupt Logic and the Config-Registers

3.2.3.1 An Example of Configuring

Configure the IT logic across Bus Q in 3 steps:

Operation	.....	#LWR	KONST1
MOV	L, IE1	#LWR	KONST2
MOV	L, IL	#LWR	KONST3
CLR	IE2	#Address	......

The KONST values are the bit samples, these bits determine that the given IT is enabled or not.

3.2.3.2 The Equations of the IT Logic Box

$$ITE = IE10 \bullet (IL0 \oplus B0) + ... + IE17 \bullet (IL7 \oplus B7) + ... + S0 \bullet IE20 + ... + TOI \bullet IE26....$$

Describing with a paralell, minterm form:

$$ITE = IE10 \bullet IL0 \bullet \overline{B0} + E10 \bullet \overline{IL0} \bullet B0 + ... + IE17 \bullet IL7 \bullet \overline{B7} + IE17 \bullet \overline{IL7} \bullet B7 +$$

$$+ S0 \bullet IE20 + ... + TOI \bullet IE26....$$

It is available to optimize it easily at transistor level too, but in our experiments, it is optimalized at gate level, because of the characteristics of the FPGA-s.

3.2.4 The Configurable Timer and Counter Modules

3.2.4.1 Block Schema and Description

This module is configurable similar as the previous one, here there are three configuration-registers. The two limiter register contains a number, which limits the counting our timing. If the counted number is greater or equal with the given TCL register, the system sends an it for the IT-logic.

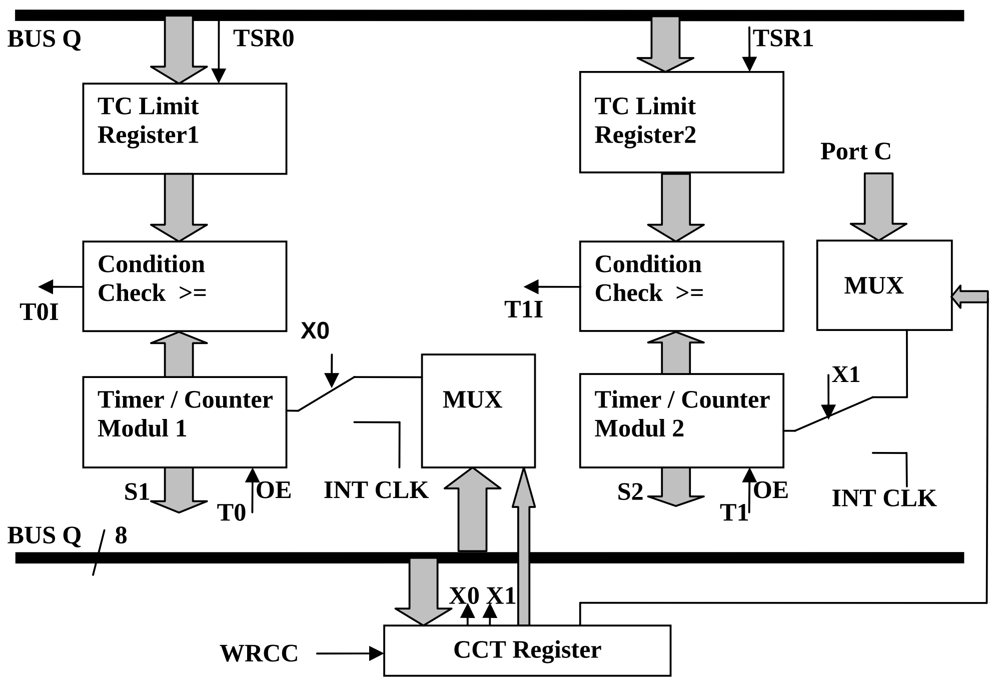

Figure 7
Configuring Registers in the Counter

The functions of the CCT register bits:

CC0(x0):	Set Timer1 mode (timing/counting)
CC1(x1):	Set Timer2 mode (timing/counting)
CC2-CC4	Select a BUS Q bit to count by Timer1.
CC5-CC7	Select a direct PortC bit to count by Timer2.

By TCM1, one of the bits of Bus Q is available to count, by TCM2 bit of PortC directly. The second case is an intelligent counting function, the processor doesn't lose time because of counting. The status of the modules can be read, at the suitable source bus, if it's necessary to perform operations with the counted number.

3.2.4.2 The Equalities of Condition Check

Bit transmission:

$$T0I_N = T0I_{N-1} \bullet (TCM_N + \overline{TCL_N}) + TCM_N \bullet \overline{TCL_N}$$

Full logic:

$$T0I = \left(\left(\left(\left(\left(\left(\left(TCM_0 + \overline{TCL_0}\right) \bullet \left(TCM_1 + \overline{TCL_1}\right) + TCM_1 \bullet \overline{TCL_1}\right) \bullet \bullet \left(TCM_2 + \overline{TCL_2}\right) + TCM_2 \bullet \overline{TCL_2}\right) \bullet \bullet (TCM_3 + \overline{TCL_3}) + TCM_3 \bullet \overline{TCL_3}\right) \bullet \bullet (TCM_4 + \overline{TCL_4}) + TCM_4 \bullet \overline{TCL4}\right) \bullet \bullet (TCM_5 + \overline{TCL_5}) + TCM_5 \bullet \overline{TCL_5}\right) \bullet \bullet (TCM_6 + \overline{TCL_6}) + TCM_6 \bullet \overline{TCL_6}\right) \bullet \bullet (TCM_7 + \overline{TCL_7}) + TCM_7 \bullet \overline{TCL_7}$$

The simple parallelization results too long forms, but by CMOS FET-s [2] with enough low saturation voltage, a low delay transistor logic is available to build much more easier.

Conclusions

Characteristics of this microcontroller solution:

Number of the program words:	65536	Rack
Program word width:	60	Bits
Size of program memory	480	kByte
Size of data memory (with 480k program memory)	64 or 32	kByte

Valuations/Comparisons:

Advantages:

- High speed of operation performing
- High operational content in one program word
- Parallelity, parallel instruction performing
- Independent, Configurable devices

Disadvantages:

- Long Program words
- Requires special Assembler and optimalizator software
- Complicated structure

Summary:

It can be seen, that by a structural reorganization it is possible to gain speed, at the expense of the growth of our program memory. With using this method, by the multiplication of the elements, more speed growth is reachable.

References

[1] Reto Zimmermann, "Computer Arithmetic: Principles, Architectures and VLSI Design", Integrated Systems Laboratory Swiss Federal Institute of Technology, 1998

[2] Ricardo Reis, Jochen A. G. Jess, "Design of System on a Chip, Devices & Components", Kluwer Academic Publishers, 2004

[3] Enoch O. Hwang, "Microprocessor Design, Principles and Practices with VHDL", Integrated Systems Laboratory Swiss Federal Institute of Technology, 2004

[4] Bob Zeidman, "An Introduction to FPGA Design", Embedded System Conference 1999

[5] Jan Gray, "Building a RISC System in an FPGA", Circuit Cellar, The Magazine for Computer Applications, March, 2000

[6] www.xilinx.com

[7] www.microchip.com

[8] www.xess.com