Generic 8-bit VHDL Processor

Introduction

This document lists out the details of a generic processor design implemented in VHDL as part of coursework on Digital Electronics Design. Consider this as a readme file for anyone trying to make sense of the design of the processor as well as its working and the associated VHDL code. In particular the following aspects of the processor are covered in this document:

Instruction Set

The following table lists the set of instructions that can be interpreted on our basic processor:

Code & List of Parameters	Operation
(ASSIGN_VALUE, k, A)	Xk <= A
(DATA_INPUT, k, j)	Xk <= INj
(DATA_OUTPUT, i, j)	OUTi <= Xj
(OUTPUT_VALUE, i, A)	OUTi <= A
(OPERATION, i, j, k, f)	Xk <= f(Xi, Xj)
(JUMP, N)	go to N
(JUMP_POS, i, N)	if Xi > 0 go to N
(JUMP_NEG, i, N)	if Xi < 0 go to N

Comments:

1. *Xn* indicates the n^{th} location in the register bank
2. *INn* indicates the n^{th} input port
3. *OUTn* indicates the n^{th} output port
4. Only 2 operations are possible through f: add/sub

Modular Components

Our processor is built with the following modules:

1. Input Selection
   Based on the instruction received, it decides from which input port data should be read from. There are 8 input ports from IN0 through IN7. Each port can read 8 bits of data and send it downstream to a specified register memory cell. It can also read input from the output of the computational resources block to store it in the register

2. Output Selection
   Selects the output port, has a similar configuration to the input ports. The outputs are registered and as such, their values remain in memory unless changed by some operation.

3. Register Bank
   The register bank stores data from the input and from the computation block before sending it to the output selection. It consists of 16 registers, X0 to X15 each capable of storing 8-bit data.

4. Computation Resources
   This is the Arithmetic unit of the processor that performs the operations described by the instruction *"f".* It consists of 8 bit adder/subtractor that computes the result of *"f"* and sends it to the input selection to store in the register.

5. Go To
   Based on the instruction received, this block can either unconditionally move to a particular instruction no. in the program memory or do a conditional jump depending on the value of a register cell.

Design Methods

In designing the processor, we can either choose to have a behavioural description of each module and accordingly code it in VHDL in the form of a Finite State Machine. The VHDL compiler can then automatically synthesise the necessary digital logic components to implement our behavioural model. This approach though simple to design and code, uses significantly less flip-flops at the cost of using more Dual Port RAMs.

Another approach for designing involves developing a structural design of each of the modules described above and then tying together those structures to make the final processor. This approach uses more flip-flops and LUTs than a behavioural design primarily because of the not-so-efficient implementation of the Register Bank.

An optimum design is achieved by having a structural description for all the components except the Register Bank. A behavioural description of the Register bank automatically creates Dual Port RAMs during synthesis and this ensures we use the lowest amount of resources for our processor.

Block Diagram

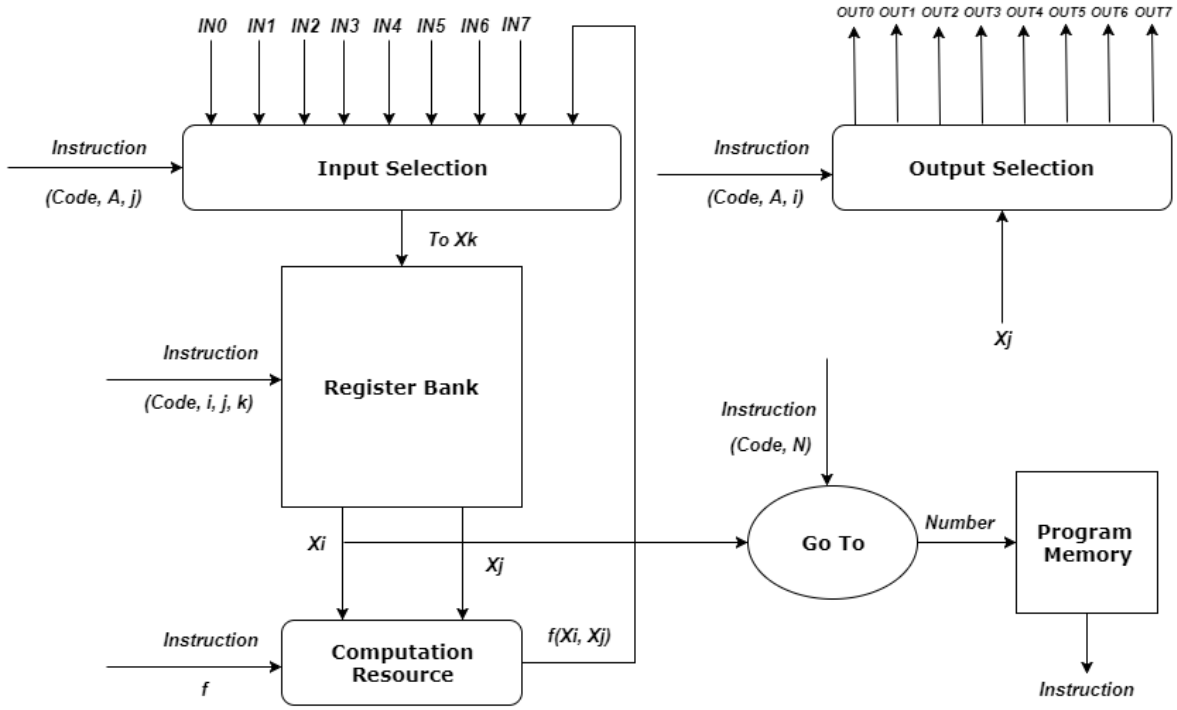

Figure 1: Block Diagram for Data Flow Graph in Generic 8-bit Processor

The above diagram lists all the components as described earlier and also shows the various input parameters that stimulate the corresponding processes for the components. The outputs and connections between various components are also indicated. An extra component not mentioned earlier is the Program Memory. This basically stores the set of instructions for performing any of the tasks that our basic processor can be programmed to perform, e.g., the chronometer program has a particular set of instructions and these are stored in the Program Memory for the processor to load the instructions through the Go To block and execute them as required.

Instruction Encoding

The program memory stores the set of instructions encoded as 16-bit vectors. Each bit in the vector is used to encode a part of the particular instruction along with the various accompanying parameters. The meaning of the bits and assignment of parameters in the 16-bit vectors (C15 down to C0) is enlisted below:

Instruction	C_{15-13}	C_{12}	C_{11-8}	C_{7-4}	C_{3-0}	Operation
ASSIGN_VALUE	000	0	A_{7-4}	A_{3-0}	k	Xk <= A
DATA_INPUT	001	0	-	j	k	Xk <= INj
DATA_OUTPUT	101	0	I	j	-	OUTi <= Xj
OUTPUT_VALUE	100	0	I	A_{7-4}	A_{3-0}	OUTi <= A
OPERATION	010	+0/-1	I	j	k	Xk <= f(Xi, Xj)
JUMP	111	0	-	N_{7-4}	N_{3-0}	go to N
JUMP_POS	110	0	I	N_{7-4}	N_{3-0}	if Xi > 0 go to N
JUMP_NEG	110	1	I	N_{7-4}	N_{3-0}	if Xi < 0 go to N

References:

[Digital Systems: From Logic Gates to Processors]

By Universitat Autònoma de Barcelona on Coursera