Floating Point Arithmetic Unit

Arnab Das, Chandrasekhar Nagarajan, Anirban Saha and Anirban Nag

***Abstract*— Many scientific applications and multimedia operations require large range of numbers and can do away with precision. Floating point numbers are used to represent long range of numbers with separate bits for exponent and mantissa. Even though this format enter numeric stability problems, they provide a large range for imprecise computation. A 16-bit floating point unit was designed in AMI 0.5µm CMOS process supporting three major operations: Addition, Subtraction & Multiplication, handling all five exceptions and operations on denormal numbers. The architecture throughput was increased, with the top module capable of servicing addition/subtraction and multiplication in parallel. Debug modes were added to get internal state bits to test the fabricated chip.**

***Index Terms*--adder, floating-point, booth's multiplier, exceptions**

I. Introduction

THE idea of this project is to implement a 16-bit floating point arithmetic unit (FPU) which allows three types of floating point operations – addition, subtraction & multiplication. The initial design was that of a 32-bit FPU, which was compliant to the IEEE 754 floating point standards. However, due to area constraint, it had to be scaled down to 16-bits, with 8 bits used for exponent, 7 bit for mantissa and 1 bit for the sign.

The unit is capable of parallel computation of multiplication and addition. The adder part of the multiplier unit reuses the adder from the floating point adder unit, resulting in reduction in hardware space.

This document provides a brief technical summary of the design considerations and testing. The remainder of the paper is organized as follows. In Section II, the design approach is discussed. Subsequently, the implementation of the individual modules is described in detail in section III. In section IV, the results obtained from simulations are discussed. In section V, the test-setup is described. Finally, in section VI, conclusions are drawn along with scope for improvement of the current work.

II. Design Approach

The proposed floating point unit was implemented in synthesizable Verilog/System-Verilog, and synthesized using existing standard cell library for Group 5 in AMI 0.5µm CMOS process technology. Based on first cut results of area/power/timing from synthesis, standard cell for 1-bit full adder was developed for optimization. There are separate state machines controlling the data flow and control once the operands are directed to their corresponding functional units.

For each operation, the following rounding mode was supported: round to nearest even. The concepts of parallelism & module sharing implemented in the FPU design are illustrated in Fig. 1. The input and output signals shown for each module will be explained in detail in the next section.

The TOP CONTROLLER is the module which integrates all submodules associated with this design. Some key design highlights of the TOP CONTROLLER is that it arbitrates the request, data and result from the 9 bit ADDER MODULE with both the MULTIPLIER CONTROL and the ADDER CONTROL.

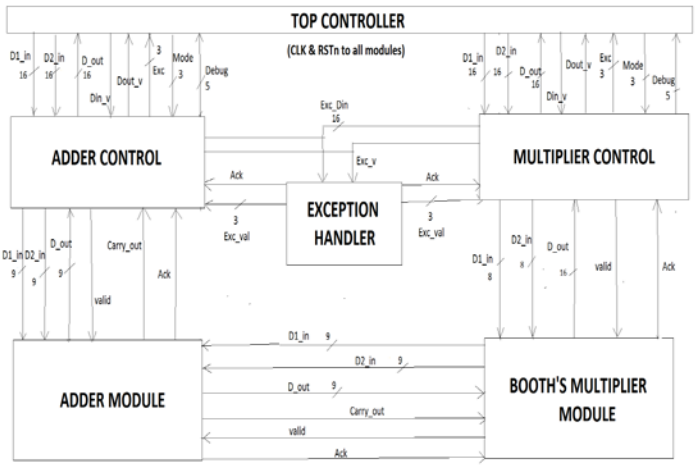

Fig. 1. Parallelism & Module sharing

The storage layout for the 16-bit FPU design is shown in Fig. 2. The first 7 LSB bits (Bit<6:0>) are used to represent the mantissa part. The exponent part is expressed by the next higher 8 bits (Bit<14:7>). Finally, the sign is represented by the MSB (Bit<15>).

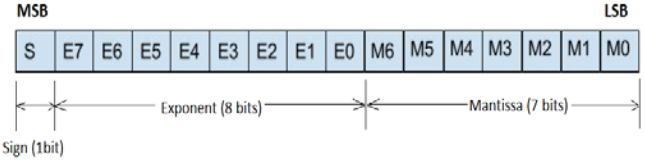

Fig. 2. Storage Layout for the 16-bit FPU

A. Das is with University of Utah, Salt Lake City, UT 84102, USA (e-mail: arnabd@utah.edu).

C. Nagarajan is with University of Utah, Salt Lake City, UT 84102, USA (e-mail: chandru@cs.utah.edu).

A. Saha is with University of Utah, Salt Lake City, UT 84102, USA (e-mail: anirban.saha@utah.edu).

A. Nag is with University of Utah, Salt Lake City, UT 84102, USA (e-mail: anirban@cs.utah.edu).

The possibility of scaling the designed FPU is shown in Fig. 3. The CORE is the host which can operate with the different FPU units. The DR signal is handy here as once the data is available in the bus, the host is intimated with which it can decide (based on its current state) when the data can be latched in. This is a good design when it comes to parallel processing of data (if the core uses a multi-threaded model).

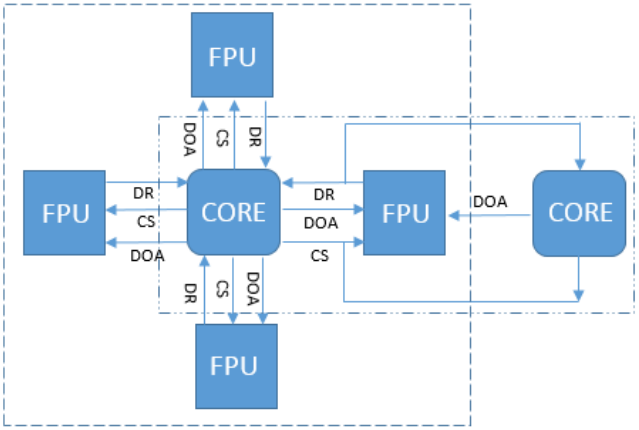

Fig. 3. Scalability

III. Implementation of Modules

The algorithm of the adder module is shown in Fig. 4. The first step involves comparing the exponents of the two numbers, and evaluating their difference (diff). Subsequently, the mantissa (or fraction) part of the number with the smaller exponent is shifted by 'diff' positions to the right. The next step involves adding the two fractions, and rounding the result to the next even (in case of our design). The exponent of the final result is the exponent of the larger number obtained in step 1, and the final mantissa is the one obtained after rounding in the previous step. The final sign bit is obtained by doing an XOR operation of the sign bits of the two operands. The resultant number is checked if it made any exceptions. In case of no exception, the number is normalized to the standard format. The normalization process makes sure that leading bit before decimal point is 1. The block diagram of the adder is shown in Fig. 5.

The algorithm for the multiplier module is shown in Fig. 6. The fraction parts of the two operands are multiplied to get the mantissa part of the final result. The exponent of the final result is obtained by adding the exponents of the two operands, and subtracting the bias value of 127 from it. The remaining operations in the multiplier module (rounding, checking for exceptions and normalization) are similar in concept to that of the adder module.

The block diagram of the multiplier unit is shown in Fig. 7. In order to save area & latency, the 9-bit adder sub-module of the adder unit was shared with the multiplier unit to perform the additions required for booth's multiplier.

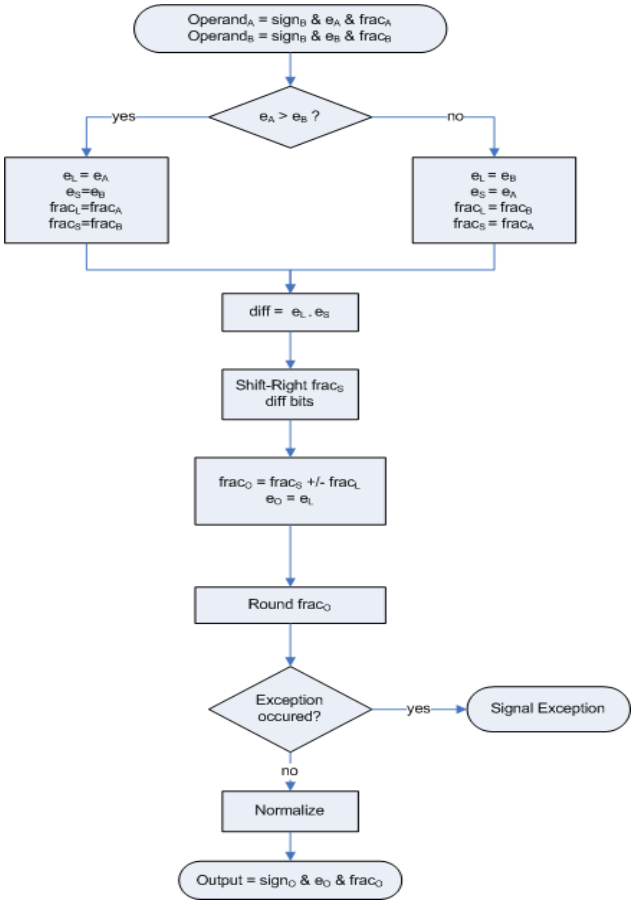

Fig. 4. Algorithm for Adder [2]

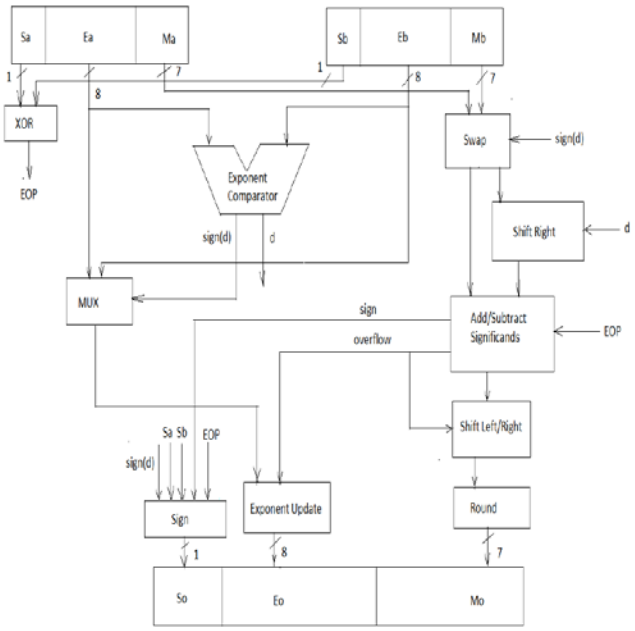

Fig. 5. Block diagram of Adder

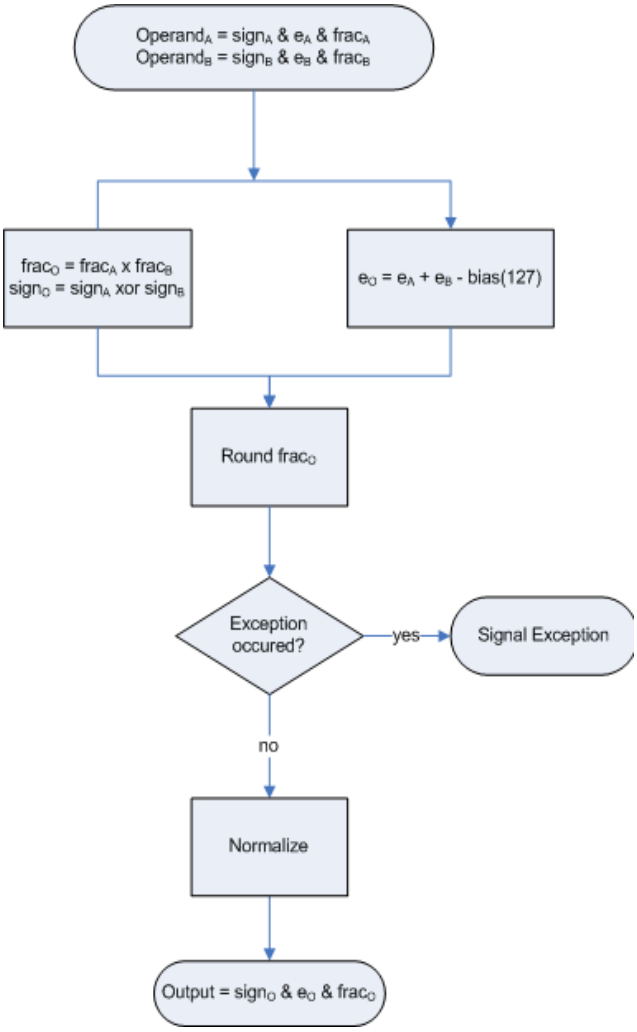

Fig. 6. Algorithm for Multiplier [2]

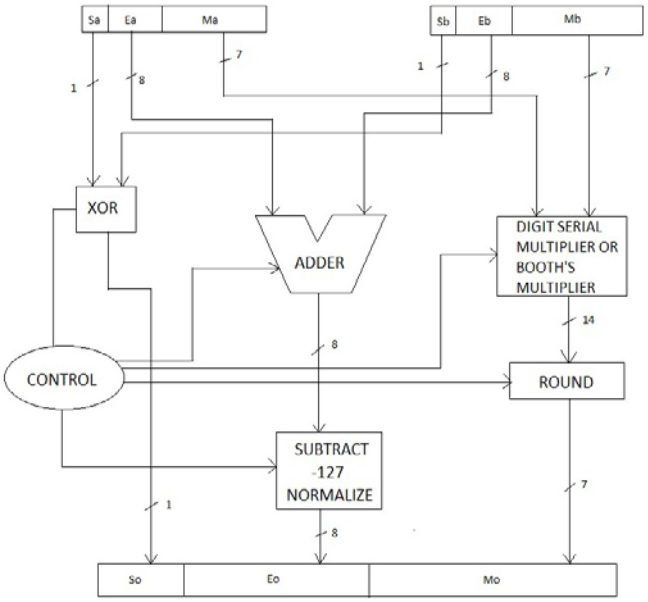

Fig. 7. Block diagram of Multiplier

The datasheet for the 16-bit FPU is shown in Table I. There is a single input line of 16 bits for supplying input operands. The two operands required for each of the operations are received sequentially. Following are the steps:

1. While in the idle/evaluation state, it loops on the Data IN Valid (DIV) signal to check for incoming data.

2. On receiving Data IN Valid, it samples the two inputs Data IN (DIN1 and DIN2) line as the first and second operands. It also samples the OPT, (0-Add, 1-Multiply) and decides on the type of operation for the received operand.

3. The operands for the adder unit and the multiplier unit cannot be interleaved. Only after one type of operands has been provided, the next type can follow.

NOTE: For the 32-bit design, in order to adhere with the pin constraints, we sampled the 32-bit input in two cycles using 32 pins (DIN) and gave the 32-bit output in 16 pins (DOUT) in two cycles. The following tricks were used:

i. On receiving Data IN Valid, it samples the corresponding input Data IN (DIN) line as the first operand. It also samples the OPT, (0-Add, 1-Multiply) and decides on the type of operation for the received operand.

ii. DIN is not necessarily required in two consecutive cycles as the DIV assertion is sufficient to indicate the continuity of the next operand. If the next operand is supplied after two cycles, it will be accepted as the second operand in its respective cycle. Note: If while providing the second operand, the operation type (OPT) is changed, it will not be latched and will adhere to the operation type defined during the latching of the first operand.

iii. The resulting data is available at Data OUT (DOUT) over 2 consecutive cycles (required by design), and valid data is indicated by the Data OUT Valid (DOV). In the first cycle internal result [31:16] and in the second cycle, result[15:0] is made available. In the first cycle itself, the corresponding busy signal (ABUSY, MBUSY) is de-asserted.

4. As soon as both operands of a particular operation have been received, the operands will be routed to the corresponding functional unit and its associated busy signal (ABUSY/MBUSY) will get asserted. The busy signal is supposed to indicate the master that the corresponding functional unit is currently in progress, and will not receive any further operational inputs.

5. However, if one unit is busy (e.g. ABUSY=1 indicating adder unit is busy but MBUSY=0), the master can provide operands for the other functional unit (that is for the multiplier unit)

6. At the completion of computation, the FPU sends the DR (data ready) signal to indicate computation results are available. The core needs to assert the chip-select CS (if not asserted) to signal the FPU to latch out the data. If the CS is already asserted, it will directly transmit the data and assert the data ready signal (DR) as well. The resulting data is available at Data OUT (DOUT) in one cycle and valid data is indicated by the Data OUT Valid (DOV). The corresponding busy signals (ABUSY, MBUSY) are de-asserted.

7. The combination of 1) De-assertion of busy and 2)

Assertion of Data OUT Valid (DOV) signals together indicates the result source being a multiplier unit output or adder unit output.

8. Exceptions like Underflow, Overflow, Not a Number (NaN), Infinity and Inexact is supported by the encoding of EXC pins. The corresponding busy signals gets de-asserted as well, but not the DOV, since the DOUT Line during an exception is illegal.

9. The MODE selects the mode of operation. MODE='000' indicates it is in normal operation mode while MODE NOT equal to '000' indicates it is in debug mode. Using the 7 non-zero MODE values, the DEBUG [4:0] pins can be configured to flush out data from the internal machine states and registers for debugging.

The diagram illustrating the debug modes is shown in Fig. 8. The FPU operates on eight different modes, specified by the Mode pins. The first mode (mode=0) is for normal operation which doesn't give any valid output on the debug pins. Mode 1 and 4 are used to show the adder module state and multiplication module state respectively in the debug pins. Mode 2 and 3 are used to output different internal bits of the adder module, while mode 5, 6 and 7 are allocated to the multiplier module for the same purpose.

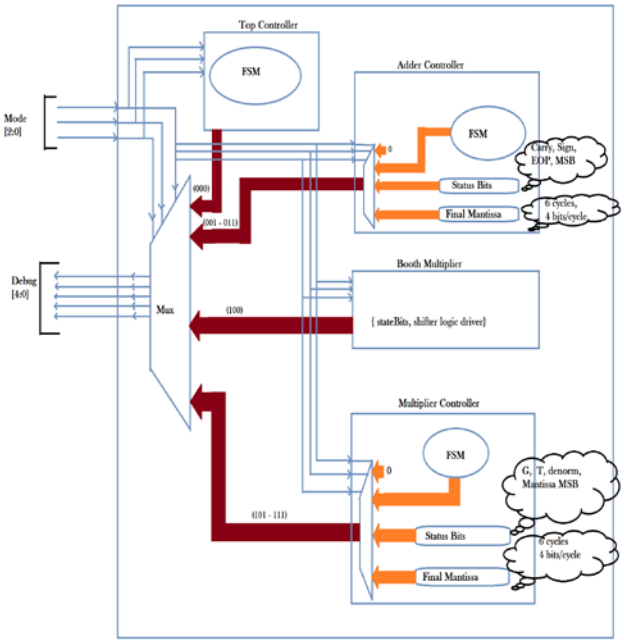

Fig. 8. Diagram showing the debug modes

IV. Results

The 32-bit FPU design implemented initially was found to be functioning correctly. The only issue with that design was that its core could not be fit into the largest 4 TCU pad frame. So, the FPU was scaled down to a 16-bit version. XXX

V. Test Setup

The test setup is shown in Fig. 9. Once the fabricated IC is received, multiple tests would be performed using either a chip tester unit or using an interface to a microcontroller. The setup can be wired using a breadboard or designing and manufacturing a custom Printed Circuit Board (PCB). The tests are conducted according to the operations described in section II. In this setup, the controller is referred to as master and the FPU is the slave.

TABLE I
DATASHEET

Pin Name	Type	Description
DIN1[15:0]	input	Data In (First Operand) 16-bits data input line for accepting 1st operands.
DIN2[15:0]	input	Data In (Second Operand) 16-bits data input line for accepting 2nd operands.
DIV	input	Data In Valid If this line is asserted, it indicates data on DataIn is valid, else the data on DataIn is invalid. This line is sampled only when either of the adder or multiplier unit is not busy, or already in the process of accepting operands
DACK	output	Data Acknowledge This is a 1 cycle pulse that indicates to the master that both the data has been latched. The master can then reuse the data lines to provide data to other functional units using chip select.
DR	output	Data Ready Indicates the master that computed data is ready for transmission. The FPU doesn't immediately transmit the data, but waits for the chip select from the master to get asserted.
DOUT [15:0]	output	Data Out 16 bits output data line.
DOV	output	Data out Valid Indicates output result on DataOut is valid data
DOA	input	Data Out Acknowledgement Acknowledgement from the master indicating the result has been latched in by the master, and the output lines can be released.
CLK	input	global clock signal for the FPU
RSTn	input	Active low Asynchronous reset. At the driver level, this can be wired to receive signal from hardware reset or software reset or both. It resets all the computation registers, guard bits to their reset values and switches back the internal state machines to Idle/Evaluation state.
CS	input	Active high Chip select: 1. This pin is used by the master to direct input data to the FPU. This provision is provided for implementations having multiple FPUs sharing common input lines. 2. This pin will also be used by the master to select the FPU for getting the result, on receiving a Data ready from the corresponding FPU.
ABUSY	output	This pin indicates the master that the adder unit is busy, hence will not be able to accept new operands for adder unit. When addition is complete, the combination of Add_busy pin (gets deasserted) and the DataOut_valid indicates to the master to sample the result as adder output.
MBUSY	output	This pin indicates the master that the multiplier unit is busy, hence will not be able to accept new operands for multiplier unit. When multiplication is complete, the combination of Mult_busy pin(gets deasserted) and the DataOut_valid indicates to the master to sample the result as multiplier output.
EXC [2:0]	output	Exceptions
OPT	input	Operation Type
MODE [2:0]	input	000: Normal operation 001-111: Debug operation
DEBUG [4:0]	output	Debug pins

Couple of test cases are:

1. Use the FPU to multiply/add

The chip select pin is pulled high, and the clock is generated for the operation. The two operands are latched into the FPU while asserting the Data In Valid (DIV) signal and having MODE [2:0] signals all pulled low. The DACK signal is monitored to ensure that both the operands are latched into the FPU. Once the DACK signal is received by the master, it pulls the CS line low. While the computation is happening, the master should monitor that the MBUSY (or ABUSY for add) line is held high. And while it is high, it should attempt to send in more values to the FPU to verify that the FPU indeed does not accept any values and does not pull the DACK line high. A point to note here is that the Exception pins (EXC [2:0]) are continuously monitored using interrupt based system by the master. Once the computation is complete, the Data Ready (DR) signal is pulled high, which should generate an interrupt in the master. In this interrupt service routine, the master should once again assert the CS line to read in the output of the computation from the DOUT [15:0]. The controller should make sure that the Data Out Valid (DOV) line is asserted at this time. Once the data is latched in by the master, it asserts the Data Out Acknowledge (DOA) to the slave to indicate the slave to release the result data from the output latch. This feature is of use when using multiple FPUs for a core. This received value is compared against what the ideal value is and the user is notified of the result.

2. Debugging

A feature of this IC is to use MODE operations to fetch internal signals for debugging purposes. The MODE [2:0] operation, if not zero is a command to the FPU to execute in debug mode. With seven different possible operations of MODE [2:0] (001-111), we enter debug mode where many internal signals can be fetched by the master at the DEBUG [4:0] lines. The definition of these signals are yet to be decided, however, the operations of data fetching from the DEBUG [4:0] lines are similar to how data is fetched from the DOUT [15:0] lines: by monitoring DR and DOV signals.

VI. Conclusion

A 16-bit FPU was designed in AMI 0.5µm CMOS process. The layout of the whole chip is shown in Fig. 10. It was possible to fit the chip core in a 4 TCU pad frame with dimensions of the core being (1893.9 µm X 1842.6 µm).

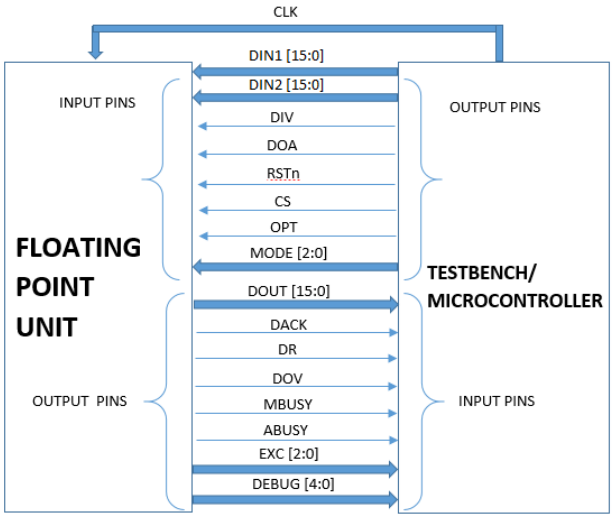

Fig. 9. Test Setup

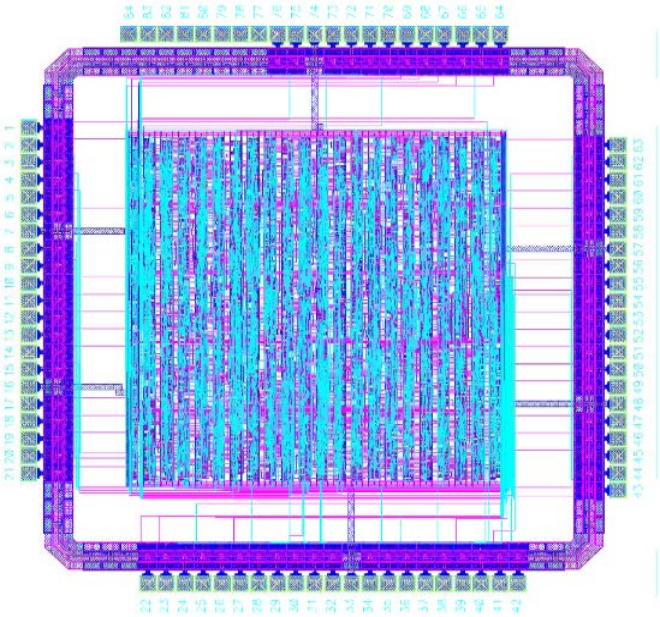

Fig. 10. Layout of the chip

Acknowledgment

The authors would like to acknowledge the continuous guidance and support of Dr. Erik Brunvand, and the lab TAs Sarvani and Daniel during the course of this design project.

References

[1] K. Karuri, R. Leupers, G. Ascheid, H. Meyr and M. Kedia, “Design and Implementation of a Modular and Portable IEEE 754 Compliant Floating-Point Unit,” in *Design, Automation & Test in Europe (DATE)*, vol. 2, pp. 1-6, 2006

[2] Jidan Al-Eryani, Floating Point Unit, opencores.org http://opencores.org/websvn,filedetails?repname=fpu100&path=%2Ffpu100%2Ftrunk%2Fdoc%2FFPU_doc.pdf