## Divider

The algorithm implemented in the original version of the processor is one of the simplest but the slowest available.  
Several other algorithms can compute the division faster but all of them present disadvantages that must be taken into account according to the target application.

Algorithms like repeated multiplication or reciprocation are fast but require a significant amount of area, similarly an array divider would have been very fast only If we had control on the place&routing process in order to create a regular structure and would have consumed a significant area: since we want to limit the area consumption we decided to implement a simple radix-4 division algorithm.  
Using an higher radix could have improved performance but the size of the look-up table required by the algorithm would have increased again the area consumption.

The divider consist in a state machine (its diagram is shown in Fig. 1) which check if the inputs will generate an overflow and performs a preliminary shift to put the divisor in the appropriate range to be computed correctly.

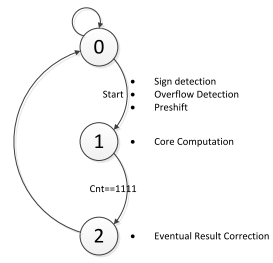
UPDATE THIS FIGURE WHEN THE DIVIDER IS FINISHED

Fig. 1 Divider State Diagram

After that, the real computation begins and lasts 16 clock cycles. The block diagram of the divider while it’s in this state is shown in Fig. 2.

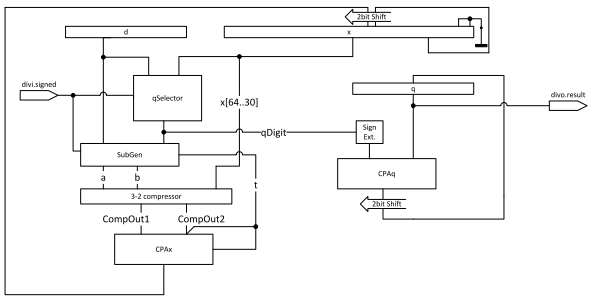


Fig. 2 Divider Block Diagram (during core computation, state=1)

The algorithm is very similar to the original radix-2 version but in this case the partial reminder (x) is shifted by 2 bits every cycle and the circuit has to guess the quotient digit from the range [-3,3]. “qSelector” is the look-up table which perform the quotient digit guessing and it’s based on the p-d plot of the radix-4 SRT division shown in Fig. 3, in case of unsigned division only the right half of the p-d plot is being used.

The quotient digits are in a radix-4 redundant format so a conversion in binary format is needed, the conversion is performed gradually every cycle by the 32-bit adder “CPAq” which shift and sum each generated digit with the already calculated quotient.

One could think that it would be better keep the quotient in a radix-4 redundant format and avoid the addition in order not to slow down the execution every cycle so that the clock frequency could be higher, but also the original divider executes a 32-bit addition every cycle so from this point of view our divider is not worse than the original one, moreover a conversion from radix-4 redundant format to binary is quite complicated, doing this it consists in a simple addition.

The same concept has been used also for the computation of the partial reminder.

An addition/subtraction in Carry-Save format would have been much faster and also easier, but the selection of the quotient digit would have required the analysis of the most significant bits of both the sum and the carry making the lookup table several orders of magnitude bigger.  
In our divider “SubGen” generates the multiple of d to sum with the current partial reminder in a carry save format, all this operands are been compressed by a 3-2 compressor (1 full adder of delay) and finally the new partial reminder is calculated with a 35-bit adder, “CPAx”.

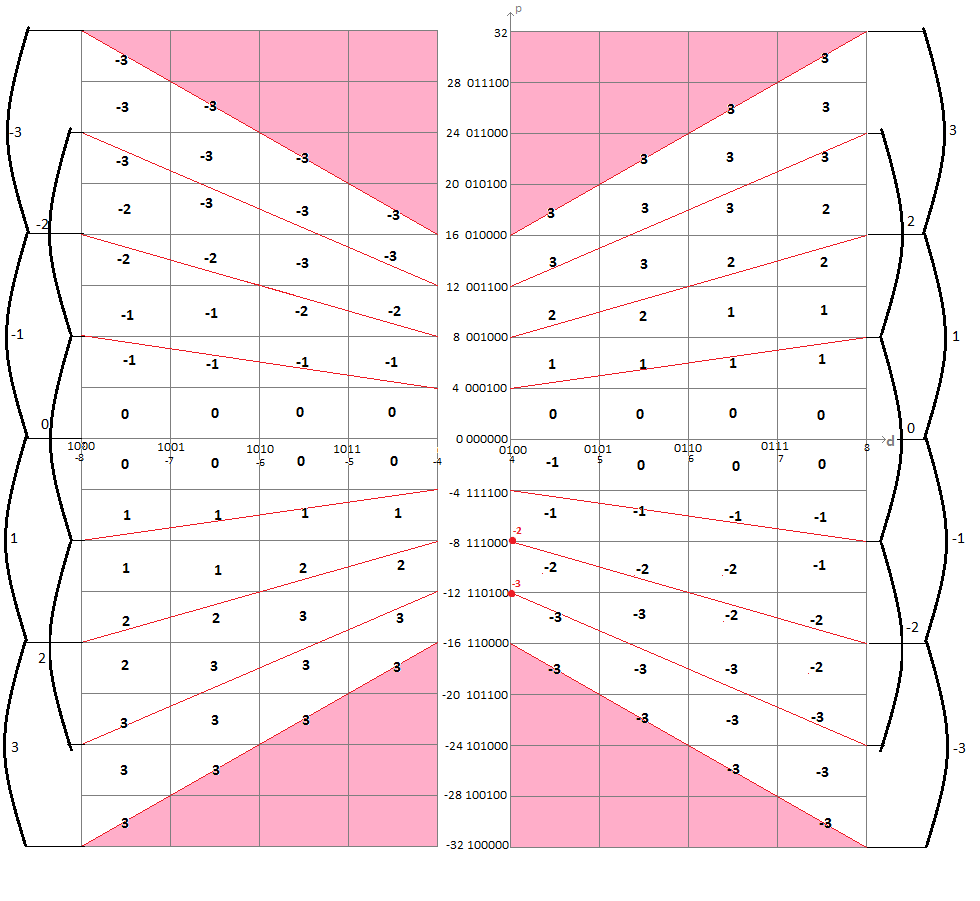


Fig. 3 Radix-4 p-d plot (red dots are exceptions)