D0 Note 3168

D Zero Central Hardware Trigger Preliminary Implementation Studies of Weighted Averaging of "Clusters".

by

R. Angstadt and M. Johnson

January 24,1997

Introduction

This is a status report on efforts to program a weighted average algorithm into an Altera Embedded Programmable Logic Device or EPLD. Input to this device will be an alternating 8 bit address in ~18.8 Ns and then an 8 bit data value in the next ~18.8 Ns (1 / 53 MHz). So every 2 * 18.8 Ns = 37.6 Ns is the maximum period we have to do any particular operation in order to keep up. Output will be an 8 bit address that is where the center of the cluster is. Altera provides a software suite of tools for a 10K series family of chips that allow programming them using language(s) and/or drawing tools. The results may be fit to a particular chip and then simulated to check the output and timing characteristics. The feasibility of implementing a weighted average algorithm was done in several partial designs to an Altera 10K50GC403-3 which has 50,000 typical gates (36,000 to 116,000 usable). This is a likely solution for this problem as the following will show. Also it is relatively cost effective at ~$75 in large quantities. This device is also reprogramable (reusable) so the algorithm may be changed (within the gate limit of the device) if a more suitable algorithm is decided upon later.

Platforms and Software Used

The Altera Software Suite "MAX+PLUS II" V7.0 and V7.1 with an Altera provided DSP V1.0 extension package to provide some canned "parameterized" mathematical symbols and functions Altera calls "megafunctions". This software will run on a variety of platforms but this work was done using Windows-95 on the Intel platform.

More About the Chips and Software

All fitting and simulation was done using the highest speed rating of the chip (the -3). And all compiling and fitting always resulted in the warning message that all timing characteristics were "preliminary" for this device. In order to interpret the results some understanding of this device is needed. These devices contains both "LAB" Logic Array Blocks (8 Elements to one block) which are configurable as standard Boolean logic elements and reconfigurable RAM which Altera names "EAB" for Embedded Array Blocks. Many "megafunctions" may be implemented in either one via menus. Each logic element is a four input user programmable gate with one output to a row or column global grid and a special fast (1 to 1.5 Ns "carry" option) . The 10K50 has 2,880 logic elements and the 10K100 has 4,992 logic elements. The 10K50 has 20,480 bits of configurable EAB (RAM). A 10K100 has 24,576 bits of configurable EAB (RAM). EAB's are very useful for making fast look up tables (LUT) if the table is below those limits. Logic Elements and RAM may be connected to each other and to I/O pins through a matrix of global row and column (static) programmable connections. Also 8 logic elements may be inter-connected on their own private busses to form a LAB. All of these resources are finite including the number of usable I/O pins which is 310 and 406 for the 10K50 and 10K100 respectively. Please see the Altera Product Literature for a more complete description and exact configuration of these devices.

The Bottom Line

It looks feasible to output a weighted average at a 25 MHz rate provided that the LUT for the divide operation can be made to fit in 20,480 bits.   Several “projects” were implemented using Altera canned library megafunctions which evaluated different aspects of the problems.   Indications are that the problem could be fit into a 10K50. “Preliminary” simulation results indicate a weighted cluster could be achieved at a 25 MHz rate. A complete design to a full specification was not achieved in the limited time available. The first somewhat simple design is instructive from the point of view of resources used but is too slow. The second more complex design remedies the speed of the first but is not fully working. The third is a quick check to make sure that the divide operation discussed in the second design meets timing requirements.

Details

The first simplistic design (Fig. 1) indicates the resources the math megafunctions use in a chip and how long the canned operations take when implemented in LAB’s. This design was called “math” which consisted of an 8 x 8 bit multiplier with a 16 bit result, one 16 bit adder which accumulated the denominator, another 8 bit adder which accumulated the numerator, two 8 bit latches, one 16 bit latch and a “dividex” megafunction which divided 16 bits by 8 to give a 16 bit integer result and 8 bit remainder. This design did give a correct weighted average result in the simulation. No comparison logic was implemented to provide cluster boundaries so it was incomplete in that way. No RAM look-up tables were done in this design; it was all fitted into Logic Elements as the Altera generated report file indicates:

Total I/O pins used:	39/304 (12%)	
Total logic cells used:	978/2880 (33%)	
Total embedded cells used:	0/80 (0%)	
Total EABs used:	0	(RAM)

(if the divide function is deleted the Logic Cell utilization becomes 529/2880=18% so this is costly function.)

Other important results are given in the “floorplan” editor which show that some resources are getting close to being used up on the “global interconnect” which is a row, half row, and channel signal routing scheme:

Row channels used	71/144	49%	
Half Row channels used	63/ 72	87%	(rows may be halved; this is the most fully utilized “Half Row”)
Columns used	12/ 24	50%	

Some Half Row interconnect resources are close to full utilization however most other resources should not be a problem. There is still room for additional necessary cluster boundary determining logic (i.e., a comparitor). There is enough room so that the dreaded “project will not fit” message should not be a problem. From the standpoint of fit this is not too bad.

Timing simulations for this design showed Addition and Multiplication could keep up (numerator out in 48.9 Ns with the denominator out (addition only) in 29.6 Ns. after start of simulation) if pipelined but that division was a real problem with the MSB coming out in 109.8 Ns and the 0th bit in 481 Ns with the last fractional bit coming out 519 Ns after the start of simulation. Clearly dividing this way is way too slow.

A second design (Fig. 2) was started, “LUT” (for Look Up Table), which would fit the dividex megafunction in a look-up table. To do this the number of bits in the look-up table must be less than the 20,480 bits available in a 10K50. An 8 bit numerator and 3 bit denominator with an 8 bit result (no fraction in this) utilized 80% (16,384 bits of the total 20,480 bits) of available EAB or RAM.   In this scheme logic was added that determined the beginning and ending cluster boundaries. Also the assumption was made that clusters are never wider than 2 bits for the address width (4 counting from 0) which according to Monte Carlo studies is true most of the time (as per Fred Borcherding). Also only the 6 MSB of the data value are used to reduce the numerator sum to 8 bits (2^6=64 max each channel for 4 channel max * 64=256= 8 bits max in the numerator). After the division is performed another stage adds back in the high order address bits to give the correct full 8 bit address. This scheme was (incompletely) implemented and fit to a 10K50.

The denominator sum is held to 3 bits (since 0+1+2+3=6 which is less than 2^3=8). The fit numbers for this design are:

Total I/O pins used:	26/304	(8%)	
Total logic cells used:	99/2880	(3%)	
Total embedded cells used:	64/80	(80%)	
Total EABs used:	8/10	(80%)	(RAM)

Row channels used	19/144	19%	
Half Row channels used	52/ 72	72%	(rows may be halved; this is the most fully utilized "Half Row")
Columns used	7/ 24	29%	

Note that the "Total Logic cells" have dropped to 99 of 2880 =3% and that the Total EAB's used have gone to 80%. I think that the Total Logic cells is too low in this design and will end up at least 12% and probably as high as 16 or 18%. Based on the first design this is about 15% too low and is indicative that something is not quite hooked up correctly so it is optimized out by the Altera compiling and fitting software. See previous D0 Note 3058 for a further discussion of this.

This design was not completed so that the simulator showed meaningful correct answers. It needs some of the latches re-arranged and a one shot so that the denominator and numerator come into the divider at the same time. In this design the LUT divider timing could not be verified with the simulator so a third minimal study was implemented as follows.

The third very brief study consisting of the dividex megafunction fit to a LUT (Fig. 3) was done last. It consists only of inputs and outputs directly to and from the 8 bit by 3 bit division scheme discussed above. The timing simulation results of this design were encouraging. The simulation results in Fig. 4 show integer division can give a result every 2 clocks with data valid for most of the half clock (9 Ns.) Dividex is not a pipelined megafunction with a data valid gate so additional logic might have to be developed to pipeline and gate this logic. Perhaps all that is necessary is a latch on the output of the divider with some timing delays for the trigger to the latch coming from the denominator adder clock out pin.

Conclusion

This looks like a technically viable scheme. Division appears to be the most time consuming and difficult operation but if the required precision can be fit into a LUT then it probably can be made to work. This study came up with a scheme to reduce required division to an 8 bit by 3 bit look up table that looks like it could be made to meet current (estimated) requirements. It may be possible to modify the included megafunction source AHDL (Altera Hardware Design Language) code to reduce the number of bits in the answer and provide a fraction although that has not yet been successful. (A very quick 10 minute attempt to modify the AHDL code failed and it was restored to it's original form by reinstalling it from the CD.)

It looks as if the other functions will be fast enough and will fit in Logic Elements in LAB's. If room is available and more speed is needed they could be implemented as a LUT in EAB's. Notes from an Altera Class indicate an 8 bit x 8 bit multiplier can be pipelined to run up to 68 MHz in an EAB. A 4 bit by 4 bit multiplier can run at 105 MHz in an EAB but only 38.9 MHz in LAB's. This indicates that LUT's implemented in EAB's are faster if the problem fits into them.

This study was done rather quickly (2 weeks total time) so only canned out of the box megafunctions were used. Other third party (expensive) megafunctions are commercially available including a floating point divide which may be faster and/or more functional but will also certainly be more expensive. Tools permit one to develop custom solutions at low or gate levels if time and expertise is available.

Further Implementation Details

The divide LUT created by the (dividex) megafunction originally created a 12 bit by 2048 bit LUT which nominally would only fit in the 24,576 bits of a 10K100 (currently $275 dollars in large quantities). The canned divide megafunctions I was using did not allow independent specification of the number of bits in the result which is automatically taken to be the sum of the number of input bits; i.e., 8 + 3 = 11 bits in this example. However by connecting only 8 of the output bits the optimization software reduced the size of the LUT to 16,384 allowing the problem to be fit in a 10K50. (Repeated attempts to use this same trick to have 4 bits as the integer answer and 4 bits as a fraction failed for various reasons.) This software is complex and feature rich requiring time to explore the parameter space it provides. Third party solutions offer additional (expensive) parameter space to explore. Tools are there to try to develop ones own optimal solution at a low (gate) or LUT level if desired. Iterating through the potential solutions takes time.

Altera is very aggressive in this market with new devices in the pipeline including faster 3.3-Volt parts including a "EPF10K50V on a 0.35 micron triple-layer metal process available now" and a larger EPF10K250A with 250,000 gates of programmable logic available at the end of 1997. Some sizes also have a DX specification which provide for doubling a global clock (provided that the maximum frequency of the device is not exceeded) The DX part may be useful in some applications. Also they have an option on some parts for holding the global clock constant throughout the device (very low propagation delay) by using a PLL (Phase Lock Loop) to advance the clock in time as it goes through the device. Their chip architecture and software attempt to guarantee the results of the simulation although in this case we always got the "preliminary" results warning. This study did not include any of these more exotic devices.

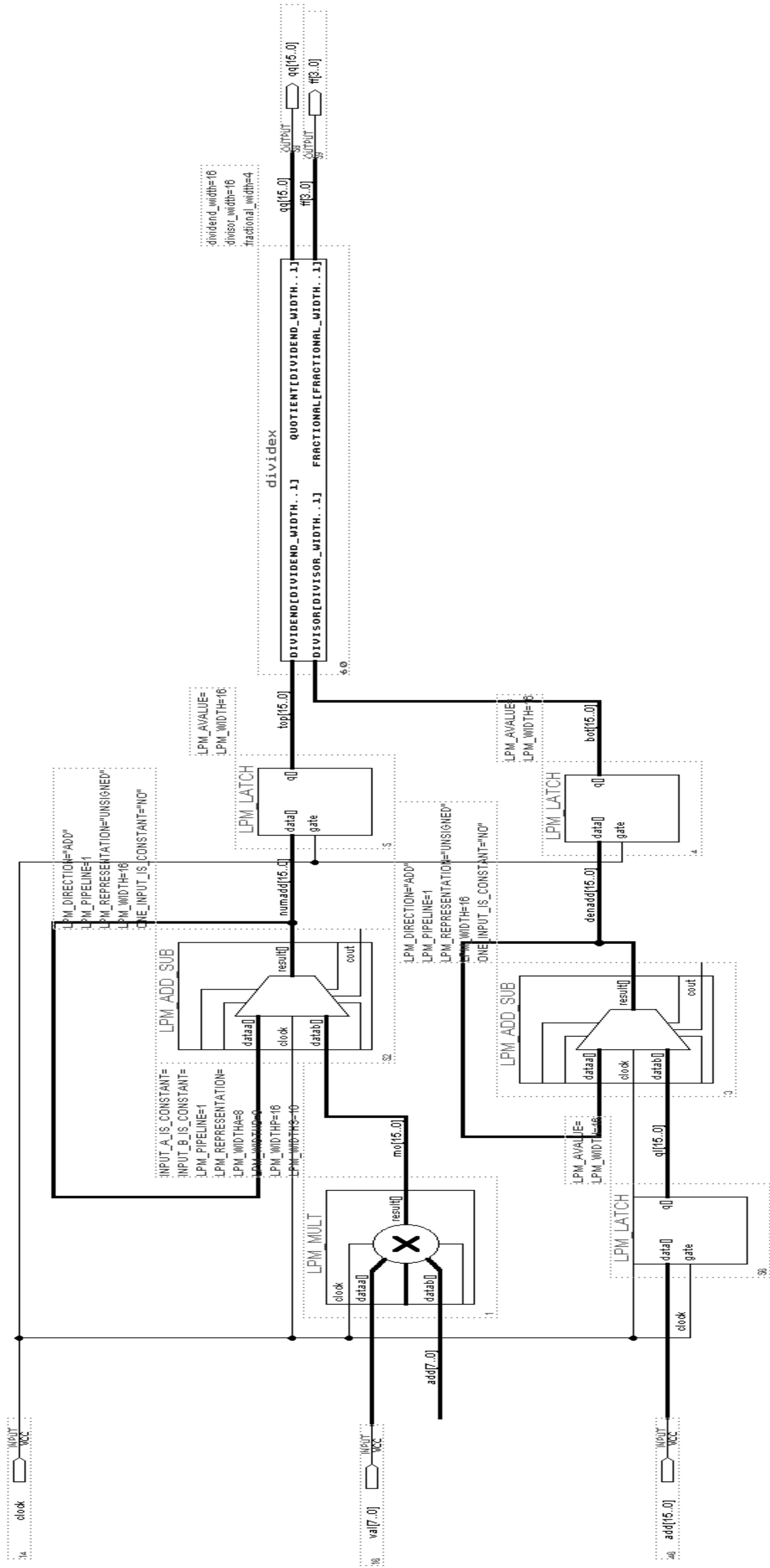
dividex
DIVIDEND[DIVIDEND_WIDTH..1]
DIVISOR[DIVISOR_WIDTH..1]
QUOTIENT[DIVIDEND_WIDTH..1]
FRACTIONAL[FRACTIONAL_WIDTH..1]
dividend_width=16
divisor_width=16
fractional_width=4
qq[15..0]
ff[3..0]
OUTPUT
top[15..0]
bot[15..0]
LPM LATCH
LPM ADD SUB
LPM MULT
numadd[15..0]
denadd[15..0]
mo[15..0]
qi[15..0]
LPM_DIRECTION="ADD"
LPM_PIPELINE=1
LPM_REPRESENTATION="UNSIGNED"
LPM_WIDTH=16
ONE_INPUT_IS_CONSTANT="NO"
clock
val[7..0]
add[7..0]
add[15..0]
INPUT
VCC

Figure # 1

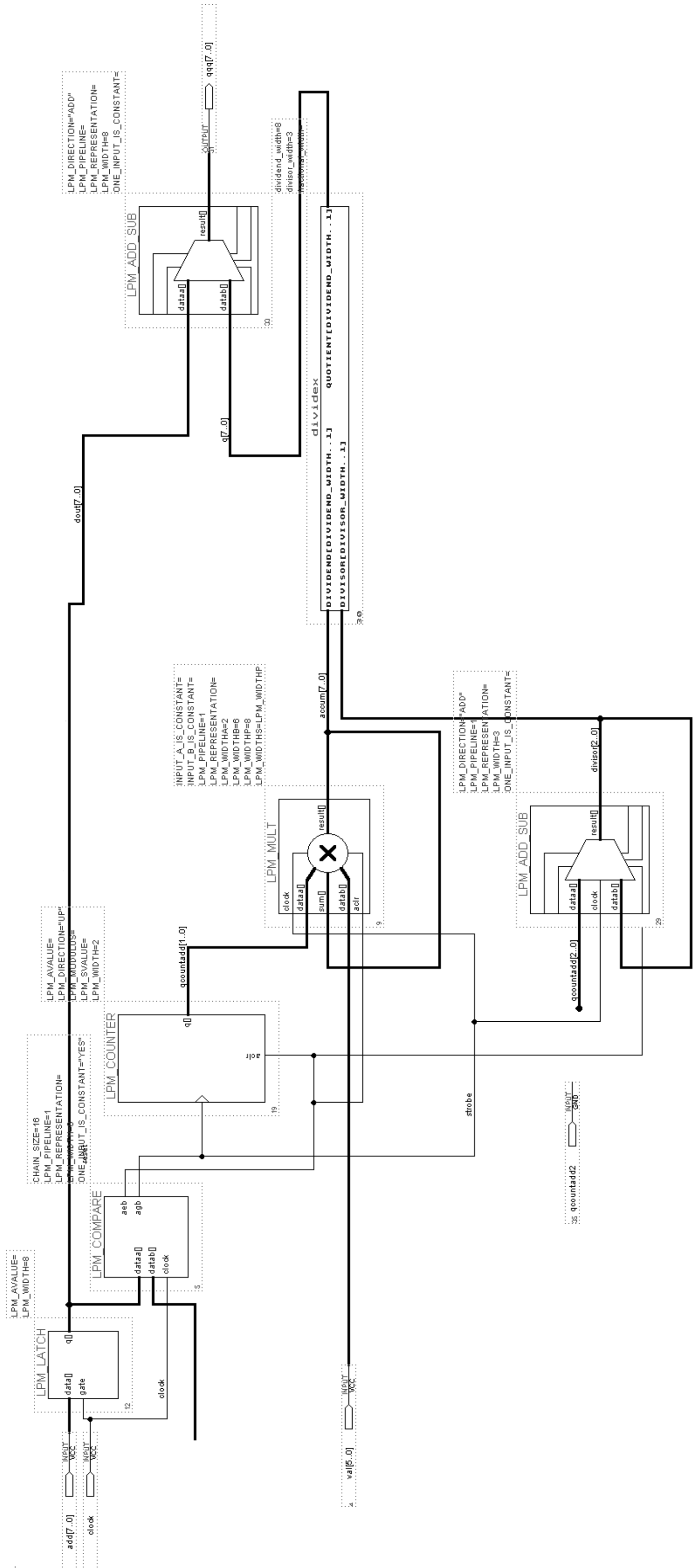
LPM_ADD_SUB
LPM_MULT
LPM_ADD_SUB
LPM_COUNTER
LPM_COMPARE
LPM_LATCH
divide
dout[7..0]
q[7..0]
qqq[7..0]
accum[7..0]
divisor[2..0]
qcountadd[1..0]
qcountadd[2..0]
strobe
clock
add[7..0]
val[5..0]
qcountadd2

Figure # 2

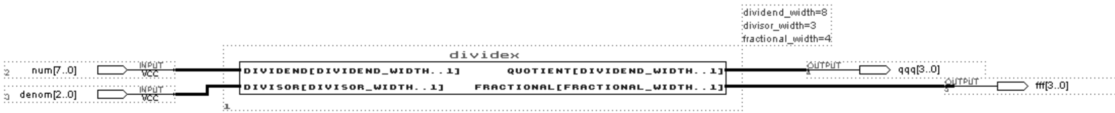
dividend_width=8
divisor_width=3
fractional_width=4
dividex
num[7..0]
INPUT
VCC
denom[2..0]
INPUT
VCC
DIVIDEND[DIVIDEND_WIDTH..1]
DIVISOR[DIVISOR_WIDTH..1]
QUOTIENT[DIVIDEND_WIDTH..1]
FRACTIONAL[FRACTIONAL_WIDTH..1]
OUTPUT
qqq[3..0]
OUTPUT
fff[3..0]

Figure # 3

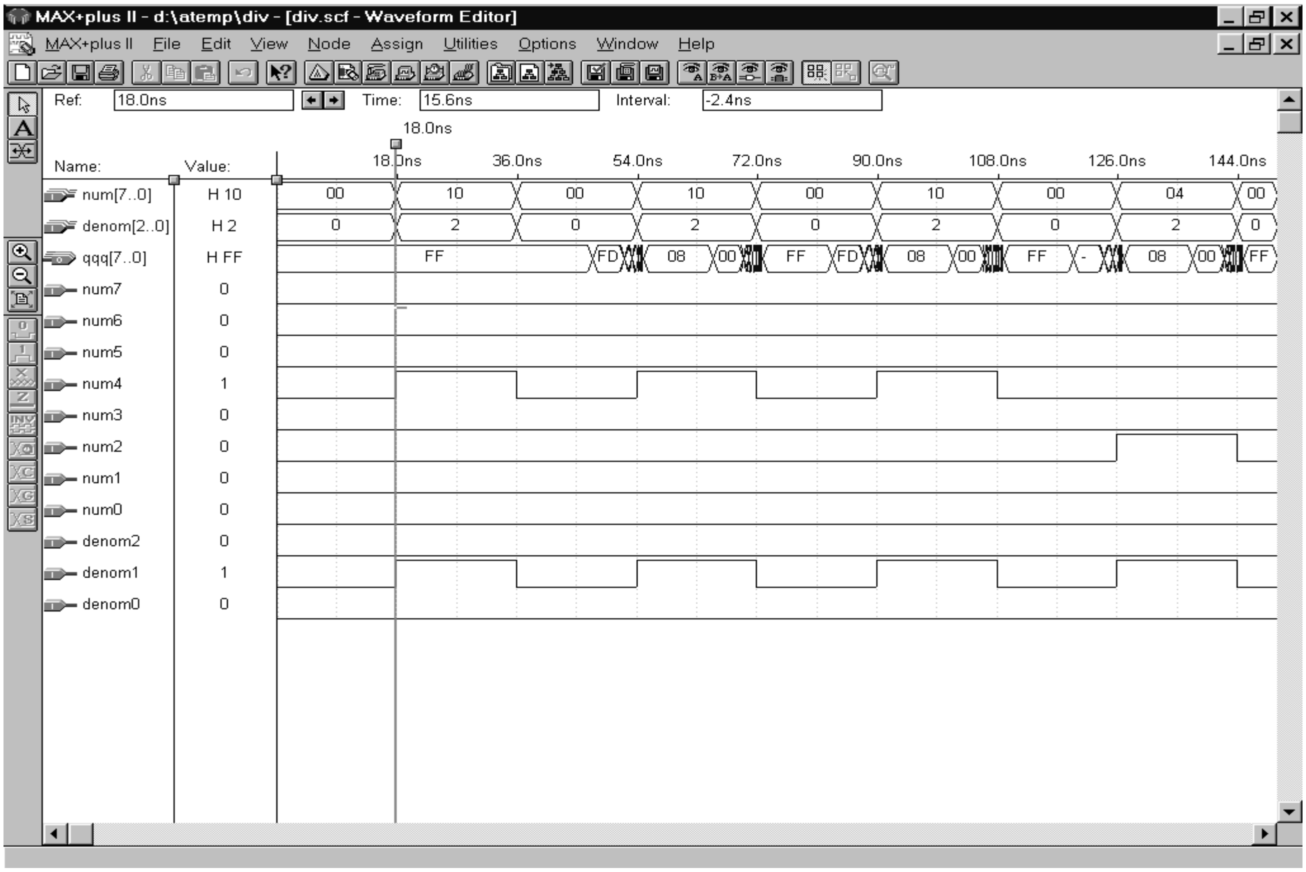
MAX+plus II - d:\atemp\div - [div.scf - Waveform Editor]
MAX+plus II File Edit View Node Assign Utilities Options Window Help
Ref: 18.0ns
Time: 15.6ns
Interval: -2.4ns
Name:
Value:
num[7..0] H 10
denom[2..0] H 2
qqq[7..0] H FF
num7 0
num6 0
num5 0
num4 1
num3 0
num2 0
num1 0
num0 0
denom2 0
denom1 1
denom0 0
18.0ns
36.0ns
54.0ns
72.0ns
90.0ns
108.0ns
126.0ns
144.0ns

Figure # 4