FT816 Floating Point Accelerator

Robert Finch – rob<remove>@finitron.ca

Overview:

FT816 floating point accelerator consists of two ninety-six bit floating point accumulators between which floating point or fixed point operations occur. Basic operations include ADD, SUB, MUL, DIV, FIX2FLT, FLT2FIX, SWAP, NEG and ABS. The floating point accumulators operate as a memory mapped device placed by default between $FEA200 and $FEA2FF. The floating point accelerator communicates through a byte wide data port and twenty-four bit address port. It was intended for use primarily with smaller byte oriented cpu's like the 65xx, 68xx series in order to provide them with some floating point capability.

Floating Point Representation:

The floating point representation is triple precision (3x a 32 bit float) and consists of a 16 bit exponent, and eighty bit mantissa. Note that the representation is a non-standard one. The mantissa is represented as a two complement number. The mantissa has one binary digit before the decimal point. The exponent is also represented as a two's complement number but with an inverted sign bit.

95 80	79 0
SEEEEEEEEEEEEEE	SM.MMMMMM.....MMMMMMMM

Range

Exponent ranges from -32768 to + 32767. The range is represented based at zero.

SEEEEEE.... field	Exponent		
FFFF	32767	maximum exponent	
...			
8000	0		
...			
0000	-32768	minimum exponent	

There are 79 bits in the mantissa plus a sign bit. So the range is -2^79 to +2^79 (approximately 24 digits of precision). The mantissa is represented in two's complement form.

Operations Supported

Floating point calculations are performed by loading the floating point accumulators with values then setting an operation code in a command register.

Operation	Opcode	
ADD	1	FAC1 = FAC2 + FAC1
SUB	2	FAC1 = FAC2 – FAC1
MUL	3	FAC1 = FAC2 * FAC1
DIV	4	FAC1 = FAC2 / FAC1
FIX2FLT	5	FAC1 = convert to float(FAC1)
FLT2FIX	6	FAC1 = convert to fixed(FAC1)
ABS	7	FAC1 = ABS(FAC1)
NEG	16	FAC1 = -FAC1
SWAP	17	FAC1 is swapped with FAC2
FIXED_ADD	81h	FAC1 = FAC1 + FAC2
FIXED_SUB	82h	FAC1 = FAC1 – FAC2
FIXED_MUL	83h	FAC1 = FAC1 * FAC2
FIXED_DIV	84h	FAC1 = FAC2 / FAC1
FIXED_ABS	87h	FAC1 = ABS(FAC1)

After the opcode is set in the command register, the operation status may be read from the status register. The most significant bit of the status register indicates a busy status.

Operation

Values are transferred to and from the FAC registers using cpu load and store instructions. Once values have been loaded into the FAC registers an operation may be performed by loading the command register with one of the given operations. Before the next operation can begin the status register must be polled to make sure that the FPU isn't busy. If the FPU is busy and another operation is specified it will be ignored.

Registers

Registers are mapped into the memory space of the system. The default is to map registers between $FEA200 and $FEA2FF. This mapping is controllable by optionally setting a parameter for the core.

$FEA200	FAC1 LSB of manitssa		
…			
$FEA209	FAC1 MSB of mantissa		
$FEA20A	FAC1 LSB of exponent		
$FEA20B	FAC1 MSB of exponent		
$FEA20F	Command / status register		
$FEA210	FAC2 LSB of manitssa		
…			
$FEA219	FAC2 MSB of mantissa		
$FEA21A	FAC2 LSB of exponent		
$FEA21B	FAC2 MSB of exponent		

Command Register

The command register is write-only and shared with the status register which is read-only. It accepts an eight bit command value. The commands supported are listed under the Operations Supported section.

Status Register

The status register located at $FEA20F has the following format:

Busy	0	0	LT	EQ	GT	ZF	VF

Busy – 1 = indicates that an FPU operation is in progress. 0 means the operation is complete.
LT – indicates that FAC1 is less than FAC2
EQ – indicates that the FAC's are equal
GT – indicates that FAC1 is greater than FAC2
ZF – indicates that FAC1 is zero (typically FAC1 holds the result of an operation)
VF – indicates that overflow occurred during the operation.

Performance

The performance of the floating point unit is at least several times what a software solution could accomplish. Performance is somewhat dependent on the data. Below is a sample.

FIX2FLT: 114 clock cycles to convert 100.0 to floating point from fixed
MUL: 176 clock cycles to multiply 100.0 * 8.0.
SUB: 33 clock cycles to subtract 100.0-8.0.
ADD: 16 clock cycles to add 100.0+8.0
DIV: 93 clock cycles to divide 100.0 / 8.0

Multiply works at a rate of one bit every two clock cycles. So it takes 160 clock cycles to process multiplication of the mantissa. There is also overhead for adjusting the sign of the operands and result.
Divide works at a rate of one bit per clock cycle. It takes 80 clock cycles to process the mantissa. There is also overhead for adjusting the signs of the operands and result.

Size

The core is estimated to be approximately 2300 4-LUTs in size (or about 2050 logic cells).

Clocks

The floating point unit uses a single clock which is also used as the clock for bus interfacing. The core may be clocked with a relatively high frequency clock.

Module Ports

module FT816Float(rst, clk, vda, rw, ad, db, rdy);

Signal	In/Out	Size	Active	Purpose
rst	I	1	high synchronous	resets the core
clk	I	1	positive edge	clocks the core
vda	I	1	high	indicates a valid data address is present
rw	I	1	high	high for read, low for write cycle
ad	I	24	high	address bus
db	I/O	8	high	bi-directional data bus
rdy	O	1	high	high when bus transfer is ready

For a write cycle the core performs a data transfer within the current clock cycle and no wait-states are incurred.

For a read cycle there are two wait-states inserted to allow the core to transfer data from internal registers, before ready becomes active.

IF the core is not addressed then the ready signal will be high allowing it to be wire-and'ed with other ready signals.

Parameters

Name	Default Value	
pIOAddress	$FEA200	This parameter controls where in the memory space the core appears. The core reserves a block of 256 consecutive addresses.
pRdyStyle	1	This parameter controls the value of the ready signal when the core is not selected. It should be 1 or 0.

Kudoos

The core originated as a direct translation of the floating point routines written in 6502 assembler code presented in Dr. Dobb's Journal, August 1976, pages 17-19. It has since been extended to higher precision and optimized for better performance. It bears little resemblance to the original code.

Floating Point Routines for the 6502

by Roy Rankin, Department of Mechanical Engineering,
Stanford University, Stanford, CA 94305
(415) 497-1822

and

Steve Wozniak, Apple Computer Company
770 Welch Road, Suite 154
Palo Alto, CA 94304
(415) 326-4248