Proc18
User's Manual

By Mike Christle

Revision History

Version	Date	Note
1.0	09/01/2020	Initial Release

Table of Contents

Introduction

My primary design goal for this project was to have a simple processor to use in FPGA designs that used minimal FPGA resources. Most FPGA manufactures provide an IP version for their products, but if I used one of these I am tied to that manufacture's FPGA. So I decided to implement my own processor, written entirely in Verilog. Now I can work with almost any FPGA. After many iterations I ended up with the Proc18 design. The 18 indicates an 18 bit instruction word, and 18 bit data.

- Integer and Boolean data types.
- Most instructions operate in a single clock cycle.
- 15 levels of interrupts.
- Precise signal timing.
- Three separate memory blocks for code, data and constants.

TERMS OF USE: MIT License

Core18

Core18 sits at the bottom of the hierarchy. Please refer to the Core18 block diagram at the end of this document. The InstDecode module decodes the instruction opcodes and generates all system timing and control signals.

The RegBank module contains 64 registers for integer data. The ALU module performs integer arithmetic. The DADRS is the address of one operand, as well as the destination address for the result. The AluSMux module selects the source of the other operand. This operand comes from the register addressed by SADRS, or the NDATA field which comes from the instruction opcode for immediate data operations. The INST input is only used for the load immediate instruction (LDI). The DATAIN input is only used for reading data from the RAM (LDR) or constants ROM (LDC).

The Status module contains two status flags Z and F. They can be set from an ALU operation, or loaded from RAM on a return from interrupt instruction (RTI). This module also sets the BRANCH signal which indicates whether a branch should be taken based on the state of the flags.

The BitMem module contains a bank of 64 Boolean registers, and the Boolean ALU to operate on them. The SADRS and DADRS input specify the source and destination addresses respectively.

The Timer module contains the timer used in the TIMER and PAUSE instructions. The TIMER instruction loads the timer value. Then the timer counts down on each clock to zero. The TIMER_ZERO signal indicates that the timer is stopped at zero. The PAUSE instruction will pause instruction execution until the timer reaches zero.

The DataMux and AdrsMux modules select the source for memory data and addresses respectively.

Proc18

Next up in the hierarchy is the Proc18 module. This includes the Core18 module and adds the three memory modules, code ROM, constants ROM, and data RAM. This module has three parameters that set the size of the three memories by setting the number of address bits for each. The code ROM can accept up to 12 address bits, for up to 4K words. The constants ROM and data RAM can accept up to 18 address bit, or up to 256K words. The DataInMux module selects the source of data driven into the Core18 module.

Simple App

At the top of the hierarchy is an application. The SimpleApp module represents a basic application that contains a periodic interrupt timer and a UART. The PortInMux and BitPaths modules are found in the Util directory, but they should always be copied to your project directory because they need to be modified for each application. The PortInMux module selects which data source drives into the System18 module.

If your application does not use any the 64 Boolean variables with discrete IO signals, only internal variables, then you don't need the BitPaths module. Just connect the BITSOUT bus around to the BITSIN bus. If you do have IO signals then add the BitPaths module and modify the source as needed.

The Decode2x4 module decodes the upper address bits and generates chip selects. The IntTimer module can be programmed to generate an interrupt signal at regular intervals, with 10uSec increments. The IntVector7 module accepts interrupt signals and encodes them into an interrupt vector. The UART module and the BaudRateClk module add a serial interface.

Build Process

I developed this project using Quartus Prime Lite Edition, and tested it on a DE10-Lite board with a MAX10 FPGA. This is the process that I use to build a project. Other tools should work in a similar fashion.

First I include all the Verilog files in the Proc18 directory, and create a symbol file for the Proc18 module. Then I include the specific Verilog files from the Util directory that I need for the project, and create a symbol file for each. Then I create a Block Diagram/Schematic File for the project. Finally I add the created symbols and wire them together as needed, see the SimpleApp diagram at the end of this document.

Assembler

This project includes an assembler written in Python3. If you would rather use a higher level language, take a look at my Miny programming language project. Miny is a high level programming language designed specifically for the Proc18 processor.

All opcodes are expressed in octal format.

Legend:
D Destination field
S Source field
P IO Port number
Z Zero flag
F Sign or shift flag
N Immediate value
A Code address

Moving Data

Syntax	Opcode	Operation	Clocks
`MOV D, S`	`41 SS DD`	`S -> D`	1
`MOV D, #N`	`61 NN DD`	`N -> D` Value must be unsigned and fit in 6 bits.	1
`LDI D, #N`	`00 07 DD` `NNNNNN`	`N -> D` This is the only 2 word instruction.	2
`LDR D, (S)`	`02 SS DD`	`RAM[S] -> D`	2
`STR (D), S`	`03 SS DD`	`S -> RAM[D]`	1
`LDC D, (S)`	`04 SS DD`	`CONST_ROM[S] -> D`	2

Integer Operations

For immediate operands the N value must be unsigned and fit in 6 bits. For all operation the Z flag is set if the result is zero. The F flag is set to the MSB of the result for all opcodes except SHR which set it to the LSB.

Syntax	Opcode	Operation	Clocks
CMP D, S	4 0 SS DD	D – S	1
NEG D, S	4 2 SS DD	-S -> D	1
INV D, S	4 3 SS DD	~S -> D	1
SHR D, S	4 4 SS DD	D >> S -> D	1
SHL D, S	4 5 SS DD	D << S -> D	1
ADD D, S	5 0 SS DD	D + S -> D	1
SUB D, S	5 1 SS DD	D - S -> D	1
MUL D, S	5 2 SS DD	D * S -> D	1
AND D, S	5 3 SS DD	D & S -> D	1
OR D, S	5 4 SS DD	D | S -> D	1
XOR D, S	5 5 SS DD	D ^ S -> D	1
CMP D, #N	6 0 NN DD	D – N	1
NEG D, #N	6 2 NN DD	-N -> D (Useless)	1
INV D, #N	6 3 NN DD	~N -> D (Useless)	1
SHR D, #N	6 4 NN DD	D >> N -> D	1
SHL D, #N	6 5 NN DD	D << N -> D	1
ADD D, #N	7 0 NN DD	D + N -> D	1
SUB D, #N	7 1 NN DD	D - N -> D	1
MUL D, #N	7 2 NN DD	D * N -> D	1
AND D, #N	7 3 NN DD	D & N -> D	1
OR D, #N	7 4 NN DD	D | N -> D	1
XOR D, #N	7 5 NN DD	D ^ N -> D	1

Boolean Operations

These instructions all operate on the 64 bits in the BitMemory. For the Bxx opcodes the Z flag is set to the value of the result, and the F flag is cleared. For the Sxx opcodes neither flag is affected.

Syntax	Opcode	Operation	Clocks
BCMP D, S	3 0 SS DD	!(D ^ S)	1
BMOV D, S	3 1 SS DD	S -> D	1
BNOT D, S	3 2 SS DD	!S -> D	1
BCLR D	3 3 SS DD	0 -> D	1
BSET D	3 4 SS DD	1 -> D	1
BAND D, S	3 5 SS DD	D & S -> D	1
BOR D, S	3 6 SS DD	D | S -> D	1
BXOR D, S	3 7 SS DD	D ^ S -> D	1
SEQ D	1 2 00 DD	Z -> D	1
SNE D	1 3 00 DD	!Z -> D	1
SLT D	1 4 00 DD	F & !Z -> D	1
SGT D	1 5 00 DD	!F & !Z -> D	1
SLE D	1 6 00 DD	F | Z -> D	1
SGE D	1 7 00 DD	!F | Z -> D	1

Calls and Jumps

When a subroutine is called, the return address is stored on the stack at the high end of RAM.

Syntax	Opcode	Description	Clocks
CALL A	0 1 AA AA	Push return address to stack, then jump to address A.	1
RTS	0 0 03 00	Jump to address popped from stack.	3
JMP A	2 0 AA AA	Always jump to address A.	2
JEQ A	2 2 AA AA	If Z jump to address A.	2
JBS A	2 2 AA AA	If Z jump to address A.	2
JNE A	2 3 AA AA	If !Z jump to address A.	2
JBC A	2 3 AA AA	If !Z jump to address A.	2
JLT A	2 4 AA AA	If F & !Z jump to address A.	2
JGT A	2 5 AA AA	If !F & !Z jump to address A.	2
JLE A	2 6 AA AA	If F | Z jump to address A.	2
JGE A	2 7 AA AA	If !F | Z jump to address A.	2

Timing

These operations are for generating short and precise delays.

Syntax	Opcode	Description	Clocks
NOP	0 0 00 00	Do nothing for one clock cycle.	1
TIMER #N	0 5 NN NN	Loads the timer with immediate value N. Then starts the counter counting down. N must be in the range 0 to 4095.	1
PAUSE	0 0 02 00	Pause execution until the timer decrements to zero. There is a 3 clock overhead.	N+3

IO Ports

Syntax	Opcode	Description	Clocks
IN D, PORT	0 6 PP DD	Input a word from an IO port.	1
OUT PORT, S	0 7 SS PP	Output a word to an IO port.	1

Interrupts

An interrupt is generated when the value of the Vector input is greater than the value of the interrupt level. The LEVEL instruction sets the interrupt level. On power-up the level is set to 15 which blocks all interrupts. Level 0 allows all interrupts. When an interrupt is generated the PC, status flags and current interrupt level are push onto the stack. Then the level is set to the value of the interrupt vector, which blocks all interrupts at that level and lower. Finally the PC is set to the value of the vector, i.e. vector 5 jumps to address 5. The first 16 locations in code ROM usually contains jump instructions to interrupt service routines. On reset the PC is set to zero, so location zero in the table should jump to the start of the application code.

Syntax	Opcode	Description	Clocks
HALT	0 0 01 00	Halt execution until an interrupt is serviced.	?
RTI	0 0 04 00	Pop PC, Z, F and LEVEL from the stack.	3
LEVEL N	0 0 05 0N	Set the interrupt level.	1

System Control

Syntax	Opcode	Description	Clocks
RESET	0 0 06 00	Pulse the RESET signal output signal. Clears bit memory and timer.	1
RESTART	2 0 00 00	Resets the program counter and stack pointer to zero.	2

Include Files

The INCLUDE directive will include another source file. All files must be in the same directory. First all files are read and written to a .tmp file. Then the .tmp file is assembled. If you only have one file, then the line number in the error messages will point to your source file. With more than one source files, the line number will refer to the .tmp file.

```
INCLUDE “afile.asm”
```

Directives

The DC directive will create a table of data in constants ROM. The label is optional. The values are a comma separated list of values.

```
LABEL DC values
```

The DS directive will reserve space in the data RAM. The label is required. The parameter indicates the number of words to reserve.

```
LABEL DS size
```

The EQU directive can be used to assign a value of an expression to a label. This label can then be used in place of a register number or a constant value. The label must be set before it is referenced. The table below lists the operators allowed in the expression in order of precedence.

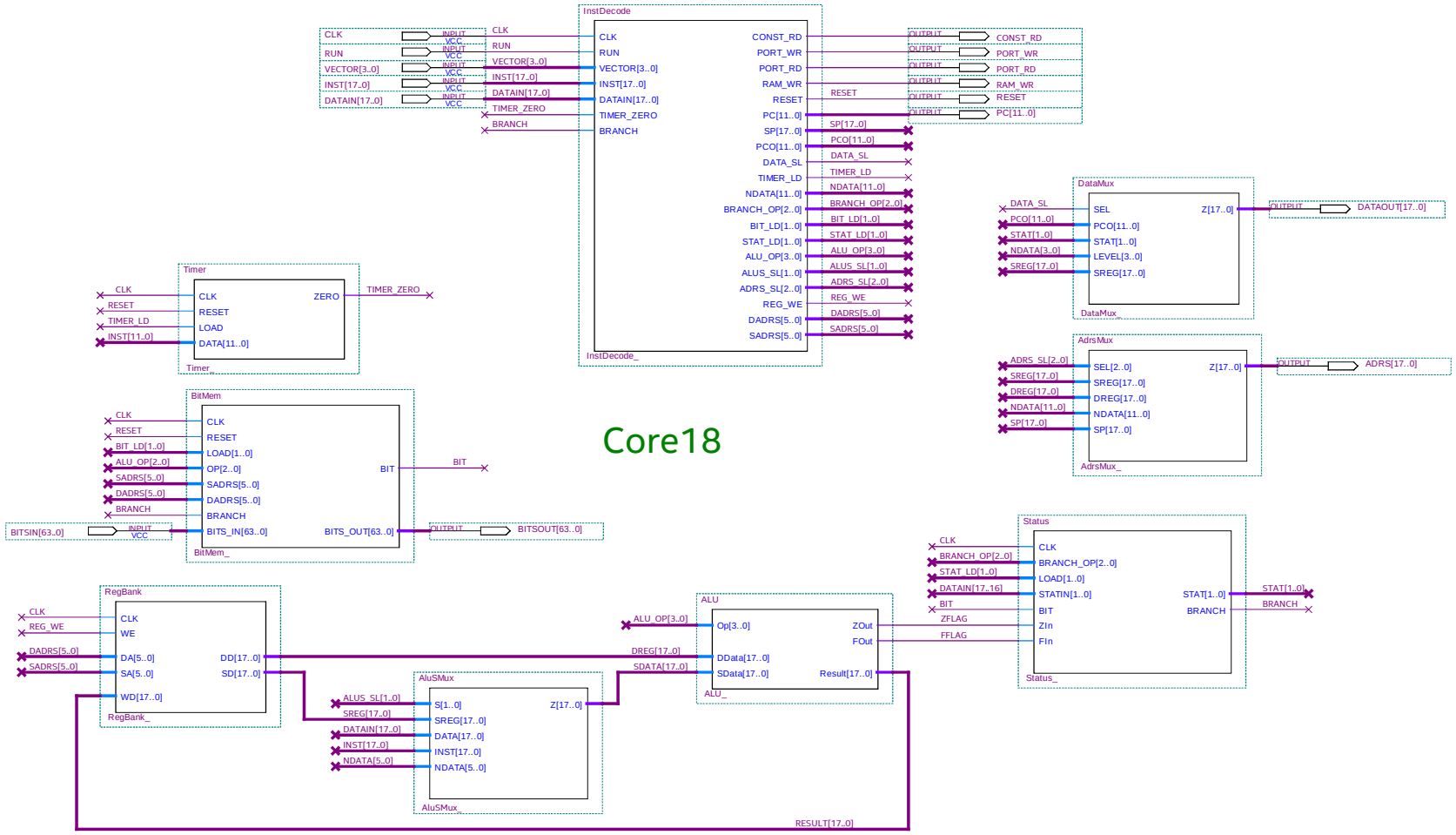
```
LABEL  EQU  expression
```

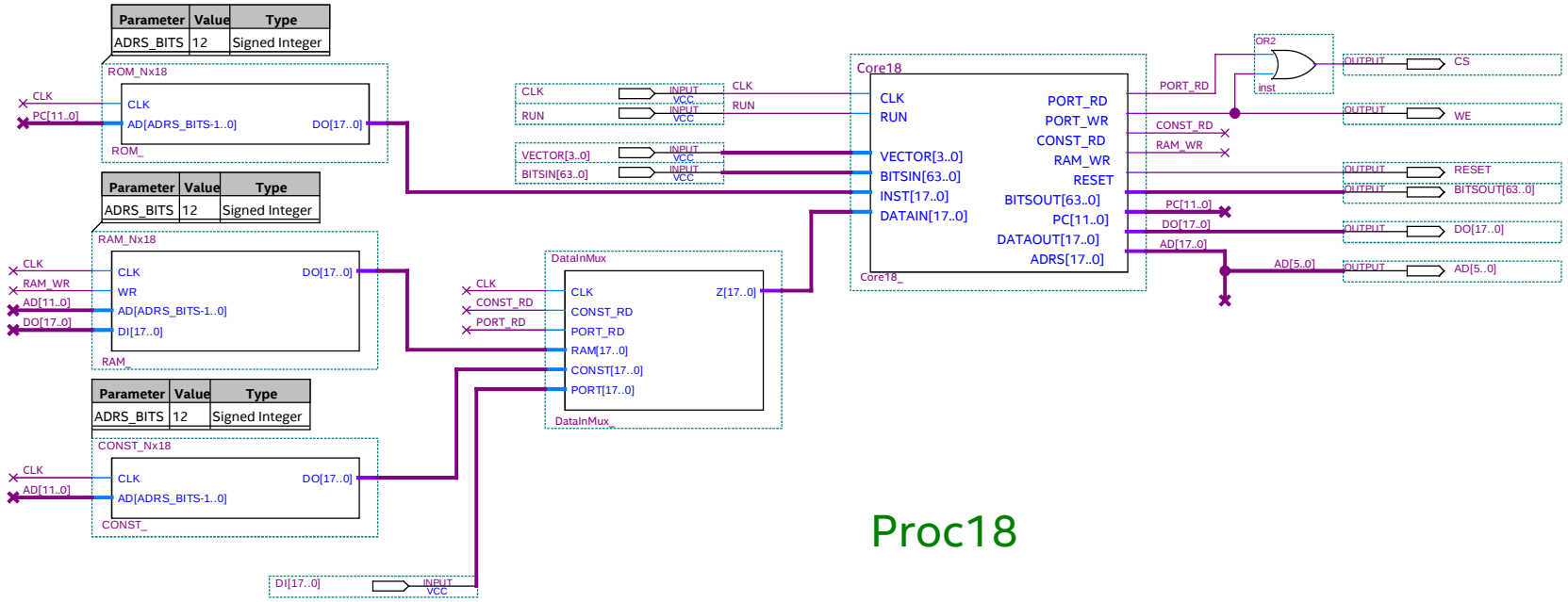
( )	Parentheses.
+ - ~	Unary positive, negative, bitwise inversion.
* / %	Multiplication, division, modulo.
+ -	Addition, subtraction.
<< >>	Shift left, shift right.
&	Bitwise AND.
^	Bitwise exclusive OR.
|	Bitwise inclusive OR.

Utility Modules

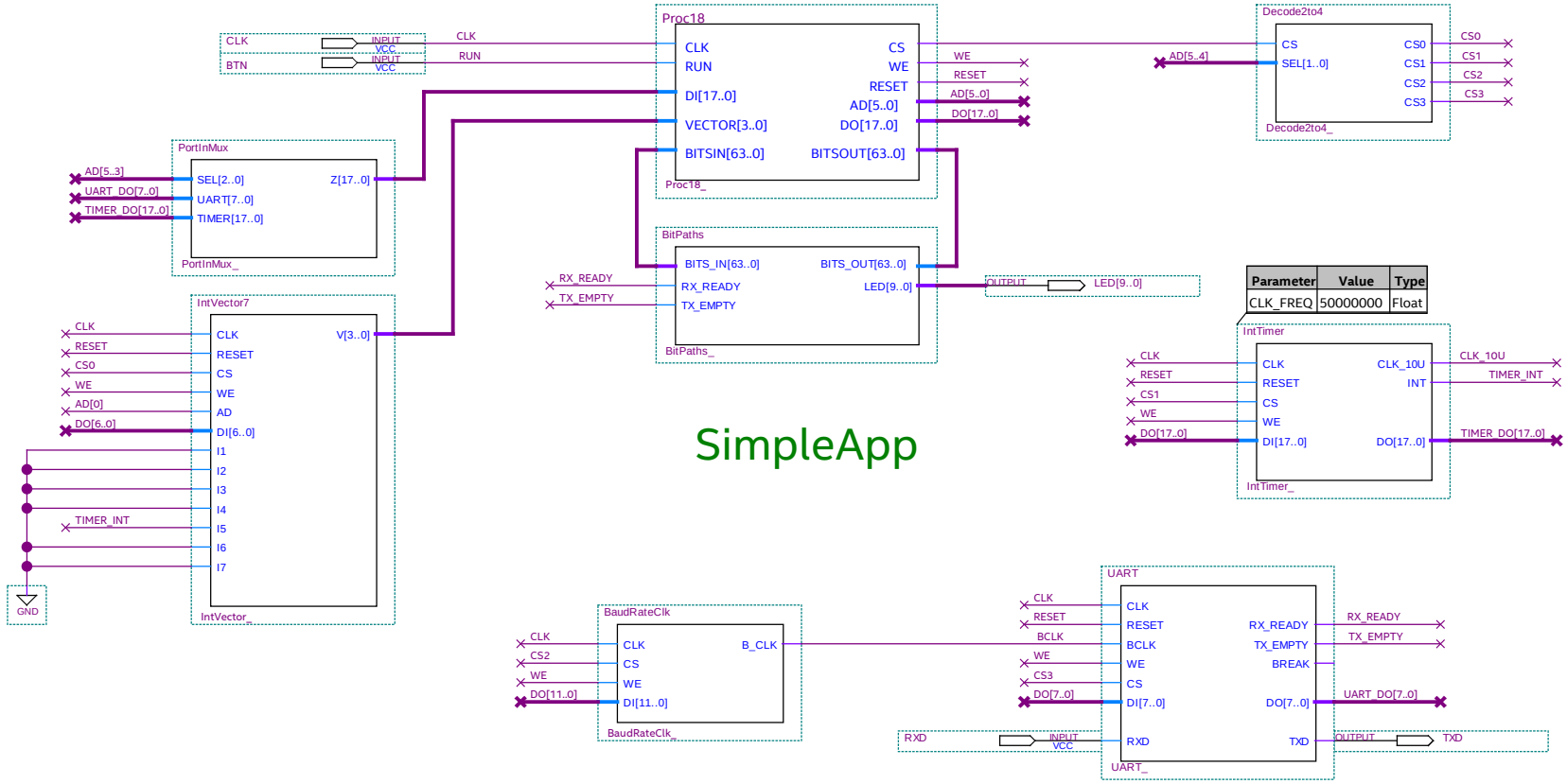
The Util directory contains a collection of utility modules that can be added to the application. The header in the Verilog source files provides implementation details.

BaudRateClk.v	Generate baud rate clock for the UART module.
BitPaths.v	Control application specific bit paths.
Decode2to4.v	Decode 2 address bits to control 4 chip selects.
Decode3to8.v	Decode 3 address bits to control 8 chip selects.
Divider.v	Signed or unsigned integer division and modulus.
I2C_Master.v	I2C master.
IntTimer.v	Periodic interrupt timer.
IntVector7.v	Mask and encode 7 interrupt sources.
IntVector15.v	Mask and encode 15 interrupt sources.
PortInMux.v	Control application specific port input paths.
SegDisplay.v	Drive up to 8 seven segment displays.
SquareRoot.v	Calculate integer square root.
UART.v	Universal asynchronous receiver transmitter.


Core18
InstDecode
InstDecode_
Timer
Timer_
BitMem
BitMem_
DataMux
DataMux_
AdrsMux
AdrsMux_
RegBank
RegBank_
ALU
ALU_
AluSMux
AluSMux_
Status
Status_
RESULT[17..0]


Parameter Value Type
ADRS_BITS 12 Signed Integer
ROM_Nx18
CLK
PC[11..0]
AD[ADRS_BITS-1..0]
DO[17..0]
ROM_
Parameter Value Type
ADRS_BITS 12 Signed Integer
RAM_Nx18
CLK
RAM_WR
AD[11..0]
DO[17..0]
WR
DI[17..0]
RAM_
Parameter Value Type
ADRS_BITS 12 Signed Integer
CONST_Nx18
CONST_
DI[17..0]
INPUT
VCC
RUN
VECTOR[3..0]
BITSIN[63..0]
DataInMux
CONST_RD
PORT_RD
RAM[17..0]
CONST[17..0]
PORT[17..0]
Z[17..0]
DataInMux_
Core18
INST[17..0]
DATAIN[17..0]
PORT_WR
BITSOUT[63..0]
DATAOUT[17..0]
ADRS[17..0]
Core18_
OR2
inst
OUTPUT
CS
WE
RESET
AD[5..0]

Proc18


SimpleApp
Proc18
Proc18_
Decode2to4
Decode2to4_
PortInMux
PortInMux_
BitPaths
BitPaths_
IntVector7
IntVector_
IntTimer
IntTimer_
BaudRateClk
BaudRateClk_
UART
UART_
Parameter
Value
Type
CLK_FREQ
50000000
Float