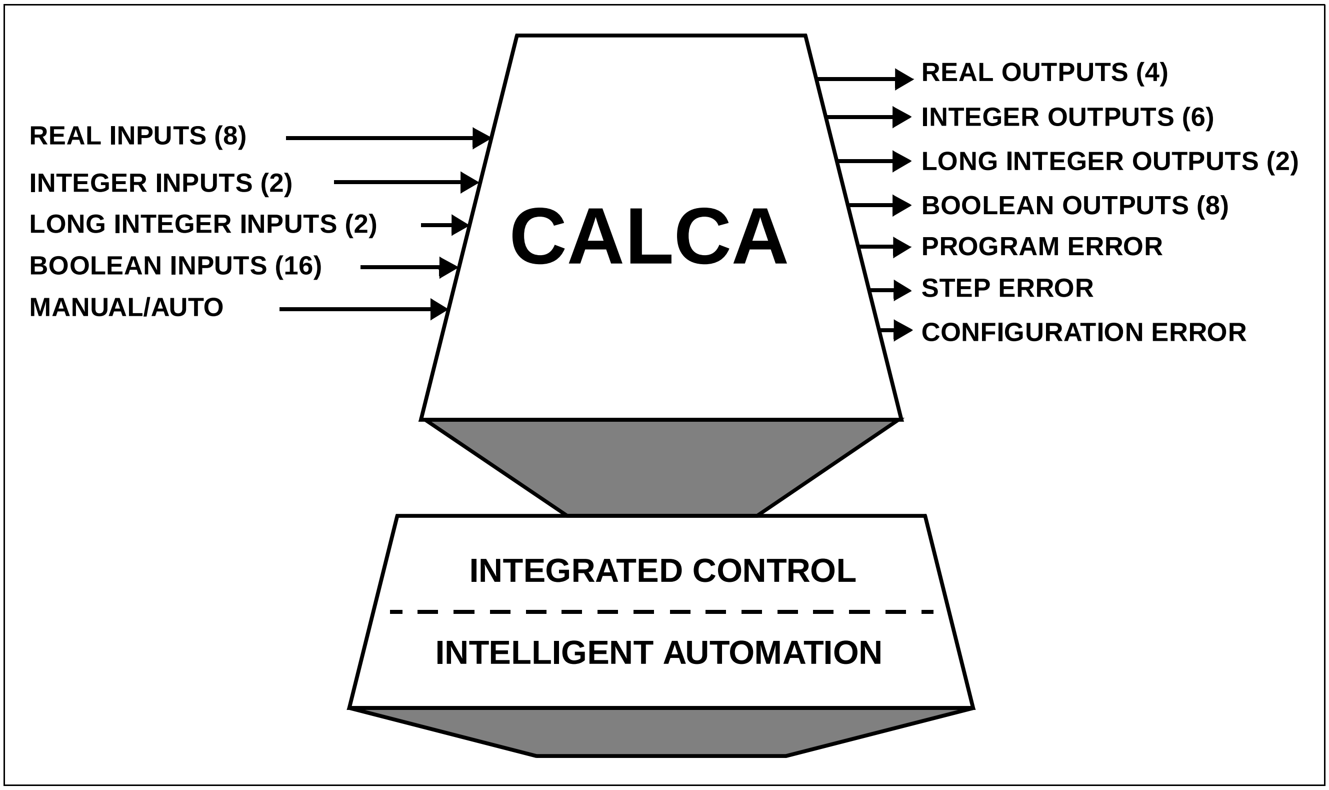
I/A Series® Software
Advanced Calculator (CALCA) Block



The CALCA block is a multiple input, 50-step, floating point, programmable calculator. It provides real-time computational capability for the modeling of specialized algorithms, signal characterization, and alteration of control waveforms to augment the operation of standard blocks.

OVERVIEW

The CALCA block provides both arithmetic and Boolean computational capability and logical functions to implement specialized control functions that cannot be implemented with either the standard control blocks or the sequence control blocks in time-critical applications.

All input connections, constant data values, and programming steps are entered via the block configuration process.

Every program step contains an *opcode*, which identifies the operation to be performed, and up to two command line arguments. The command line arguments consist of the actual operands for the step, the location of the operands, a specification of details that further refine the opcode, or some combination of these factors.

STANDARD FEATURES

- Inputs:
  - 8 real
  - 2 long integer
  - 2 integer
  - 16 boolean

- Outputs:
  - 4 real
  - 2 long integer
  - 8 boolean
  - 6 integer
- Auto/Manual control of the real outputs, which can be initiated by a host process or another block
- 24 floating point memory data storage registers that are preserved between execution cycles
- Stack of 24 floating point values for storage of intermediate computational results – provides chaining ability for up to 24 calculations
- Up to 50 programming steps of up to 16 characters each
- Initialization of all timers and memory registers
- Dual operand capability for several mathematical and logic instructions
- Conditional execution of arithmetic calculations, depending on arithmetic or logic conditions detected under program control
- Interchangeable arithmetic or Boolean operations
- Almost unlimited time delays and pulse widths in the timer instructions.
- Algorithm ability to read the status bits (for example, Bad, Out-of-Service, Error) of input/output parameters and directly control the status bits of output parameters
- Forward branching of program control
- Propagation of the cascade acknowledgment from an upstream block to a downstream block
- Propagation of cascade initialization request from a downstream block to an upstream block
- Syntax check of all programming steps during block installation and reconfiguration
- Input and output parameter error detection and control
- Detection of program runtime errors

INSTRUCTIONS

Arithmetic

ABS	Absolute value
ACOS	Arc cosine
ADD	Add
ALN	Natural antilog
ALOG	Common antilog
ASIN	Arc sine
ATAN	Arc tangent
AVE	Average
CHS	Change sign
COS	Cosine
DEC	Decrement
DIV	Divide
EXP	Exponent
IDIV	Integer division
IMOD	Integer modulus
INC	Increment
LN	Natural logarithm
LOG	Common logarithm
MAX	Maximum
MIN	Minimum
MEDN	Median
MUL	Multiply
RAND	Generate random number
RANG	Generate random number, Gaussian
RND	Round
SEED	Seed random number generator
SIN	Sine
SQR	Square
SQRT	Square root
SUB	Subtract
TAN	Tangent
TRC	Truncate

Boolean Logic

AND	Logical AND
ANDX	Packed logical AND
NAN	Logical not AND
NOR	Logical not OR
NORX	Packed logical not OR
NOT	NOT
NOTX	Packed logical NOT
NXO	Logical not exclusive OR
NXOX	Packed logical not exclusive OR
OR	Logical OR
ORX	Packed logical OR
XOR	Logical exclusive OR
XORX	Packed logical exclusive OR

Input/Output Reference

CBD	Clear bad status
CE	Clear error status
COO	Clear out-of-service status
IN	Input
INB	Input indexed Boolean
INH	Input high order
INL	Input low order
INR	Input indexed real
INS	Input status
OUT	Output
RBD	Read bad and out-of-service bits
RCL	Read and clear
RCN	Read connect status
RE	Read error bit
REL	Clear secure status
RON	Read in-service status
ROO	Read out-of-service bit
RQE	Read quality including error
RQL	Read quality
SAC	Store accumulator in output
SBD	Set bad status
SE	Set error status
SEC	Set secure status
SOO	Set out-of-service status
STH	Store high order
STL	Store low order
SWP	Swap

Cascade

PRI	Propagate upstream
PRO	Propagate downstream
PRP	Propagate errors

Memory and Stack Reference

CLA	Clear all memory registers
CLM	Clear memory register
CST	Clear stack
DUP	Duplicate
LAC	Load accumulator
LACI	Load accumulator indirect
POP	Pop stack
STM	Store memory
STMI	Store memory indirect
TSTB	Test packed Boolean

Program Control

BIF	Branch if false
BII	Branch if initializing
BIN	Branch if negative
BIP	Branch if positive or zero
BIT	Branch if true
BIZ	Branch if zero
END	End program
EXIT	Exit program
GTI	Go to indirect
GTO	Go to
NOP	No operation

Clear/Set

CLR	Clear
CLRB	Clear packed Boolean
SET	Set
SETB	Set packed Boolean
SSF	Set and skip if false
SSI	Set and skip if initializing
SSN	Set and skip if negative
SSP	Set and skip if positive
SST	Set and skip if true
SSZ	Set and skip if zero

Timing

CHI	Clear history
CHN	Clear step history
DOFF	Delayed OFF
DON	Delayed ON
OSP	One-shot pulse
TIM	Time since midnight

Logic

FF	Flip-flop
MRS	Master reset flip-flop

Error Control

CLE	Clear error
RER	Read error
SIEC	Skip if error clear

EXAMPLES

Figure 1 shows a program example that includes a typical instruction (ADD) which uses two inputs (dyadic). Figure 2 shows the stack operation for each program instruction in Figure 1. Figure 3 shows a program example that includes a typical instruction (AVE) which uses more than two inputs (polyadic). Figure 4 shows the stack operation for each program instruction in Figure 3. Figure 5 shows a program branching example. Figure 8 shows the timing diagram for a program example using the DON timing instruction.

STEP01	ADD RI01 RI02	Adds RI01 to RI02 and pushes the result (Sum1) onto stack
STEP02	ADD RI03 RI04	Adds RI03 to RI04 and pushes the result (Sum2) onto stack
STEP03	ADD	Pops Sum2 and Sum1 from stack, performs addition, and pushes the result (Sum3) onto stack
STEP04	IN 4	Pushes constant “4” onto stack
STEP05	DIV	Pops ‘4’ and Sum3 from stack, divides them, and pushes Quotient onto stack

Figure 1. Program Example with Typical Dyadic Instructions

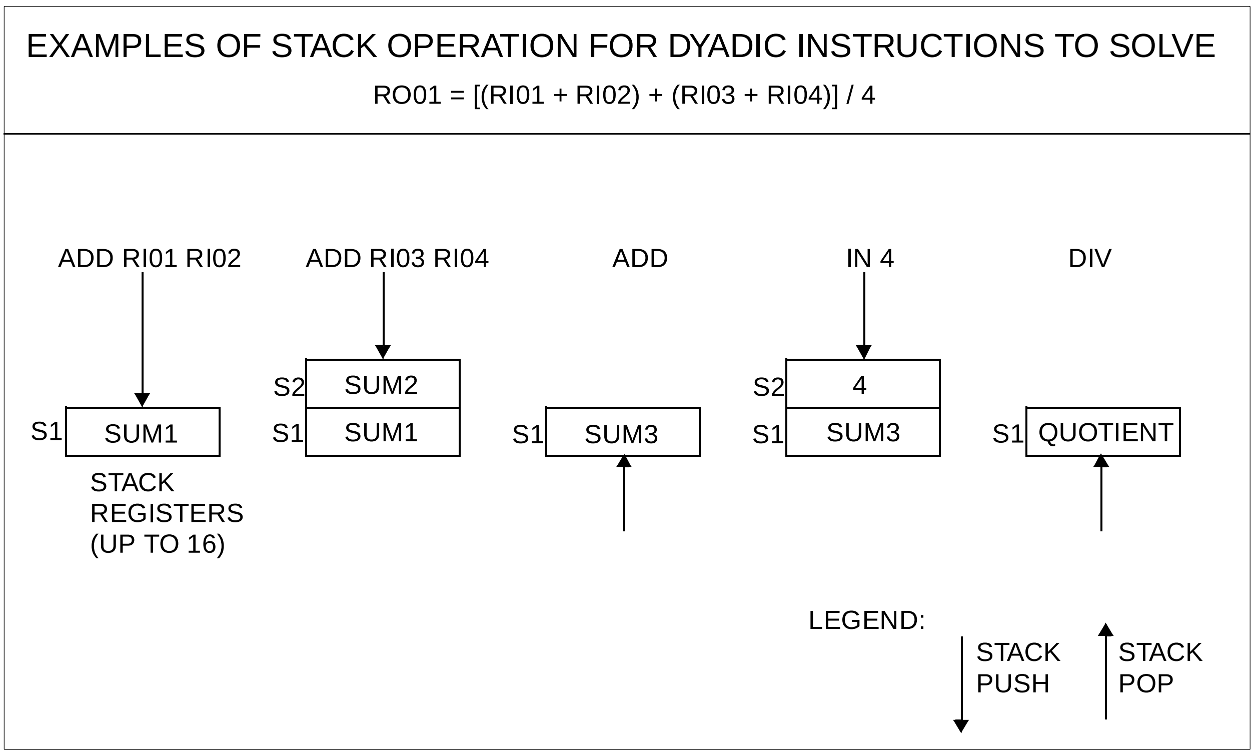


Figure 2. Examples of Stack Operation for Dyadic Instructions

STEP01	CST	Clears stack
STEP02	IN RI01	Pushes RI01 value onto stack
STEP03	IN RI02	Pushes RI02 value onto stack
STEP04	IN RI03	Pushes RI03 value onto stack
STEP05	IN RI04	Pushes RI04 value onto stack
STEP06	AVE	Pops Value4 to Value1 from stack, averages them, and pushes Average onto stack

Figure 3. Program Example with Typical Polyadic Instruction (AVE)

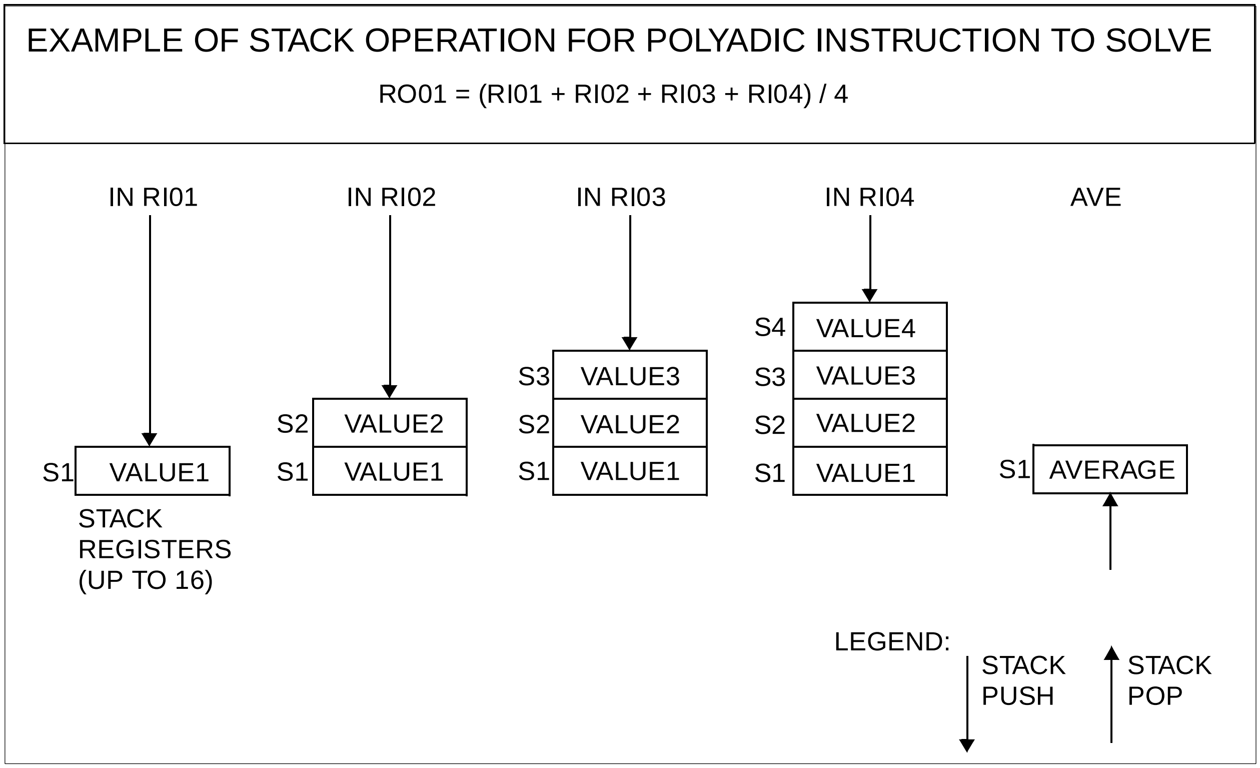


Figure 4. Examples of Stack Operation for Polyadic Instruction

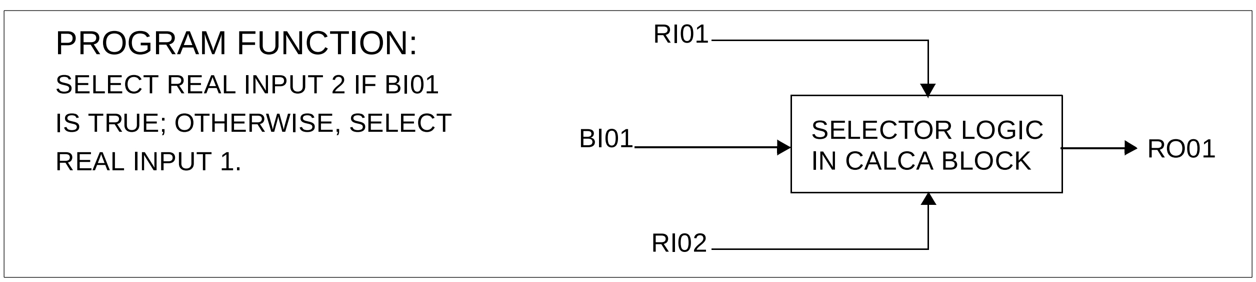


Figure 5. Program Branching Diagram

STEP01	IN BI01	Reads Boolean input 1
STEP02	BIT 05	Branches to Step 5 if BI01 is true
STEP03	IN RI01	Reads Real Input 1
STEP04	GTO 06	Branches to Step 6
STEP05	IN RI02	Reads Real Input 2
STEP06	OUT RO01	Writes selected real value to output

Figure 6. Program Example with Branching Instructions

STEP01	IN BI01	Inputs the value of Boolean input 1 to the accumulator each time the block executes
STEP02	DON 7	If BI01 remains true for 7 seconds, DON writes a 1 to the accumulator; otherwise, it writes a 0 to the accumulator
STEP03	OUT BO03	Outputs accumulator contents to Boolean output BO03

Figure 7. Program Example with DON Timing Instruction

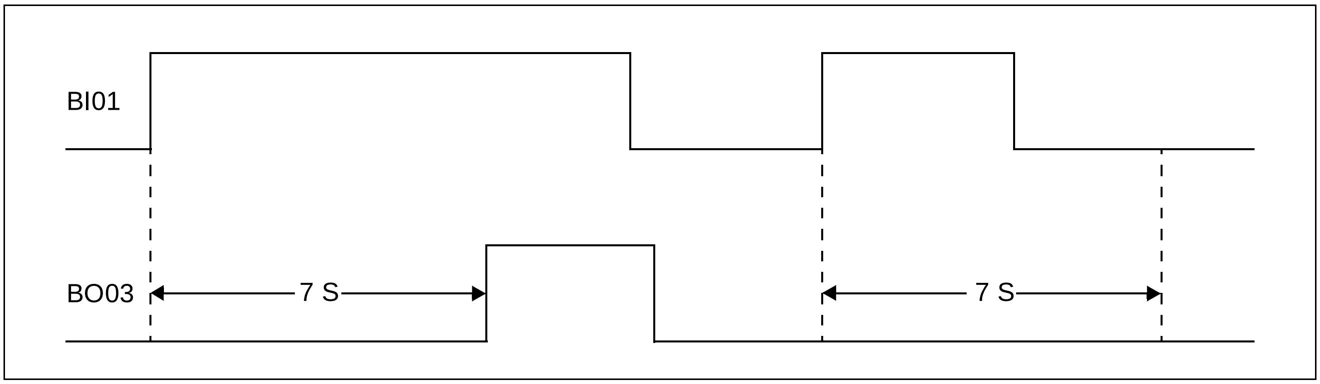


Figure 8. Timing Diagram for DON Example

The Foxboro Company
33 Commercial Street
Foxboro, Massachusetts 02035-2099
United States of America
http://www.foxboro.com
Inside U.S.: 1-508-543-8750 or 1-888-FOXBORO (1-888-369-2676)
Outside U.S.: Contact your local Foxboro Representative.

MB 021 Printed in U.S.A. 0896

An Invensys company