FUNCTION\_BLOCK STACK\_INT

VAR\_INPUT PUSH, POP: BOOL R\_EDGE; (\* Basic stack operations \*)

R1 : BOOL ; (\* Over-riding reset \*)

IN : INT ; (\* Input to be pushed \*)

N : INT ; (\* Maximum depth after reset \*)

END\_VAR

VAR\_OUTPUT EMPTY : BOOL := 1 ; (\* Stack empty \*)

OFLO : BOOL := 0 ; (\* Stack overflow \*)

OUT : INT := 0 ; (\* Top of stack data \*)

END\_VAR

VAR STK : ARRAY[0..127] OF INT; (\* Internal stack \*)

NI : INT :=128 ; (\* Storage for N upon reset \*)

PTR : INT := -1 ; (\* Stack pointer \*)

END\_VAR

(\* Function Block body \*)

LD R1 (\* Dispatch on operations \*)

JMPC RESET

LD POP

ANDN EMPTY (\* Don't pop empty stack \*)

JMPC POP\_STK

LD PUSH

ANDN OFLO (\* Don't push overflowed stack \*)

JMPC PUSH\_STK

RET (\* Return if no operations active \*)

RESET: LD 0 (\* Stack reset operations \*)

ST OFLO

LD 1

ST EMPTY

LD -1

ST PTR

CAL LIMIT(MN:=1,IN:=N,MX:=128)

ST NI

JMP ZRO\_OUT

POP\_STK: LD 0

ST OFLO (\* Popped stack is not overflowing \*)

LD PTR

SUB 1

ST PTR

LT 0 (\* Empty when PTR < 0 \*)

ST EMPTY

JMPC ZRO\_OUT

LD STK[PTR]

JMP SET\_OUT

PUSH\_STK: LD 0

ST EMPTY (\* Pushed stack is not empty \*)

LD PTR

ADD 1

ST PTR

EQ NI (\* Overflow when PTR = NI \*)

ST OFLO

JMPC ZRO\_OUT

LD IN

ST STK[PTR] (\* Push IN onto STK \*)

JMP SET\_OUT

ZRO\_OUT: LD 0 (\* OUT=0 for EMPTY or OFLO \*)

SET\_OUT: ST OUT

END\_FUNCTION\_BLOCK