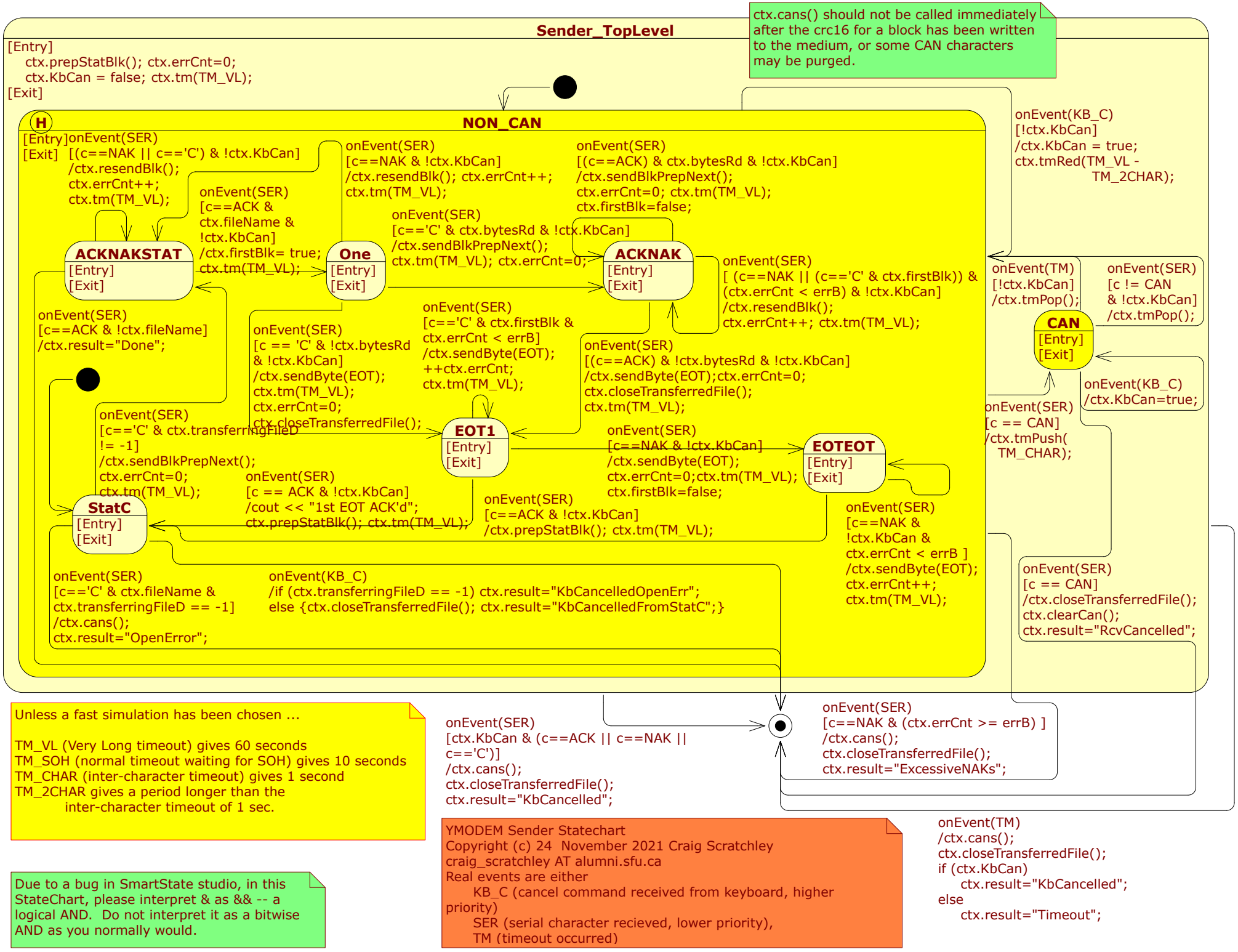
Sender_TopLevel
[Entry]
ctx.prepStatBlk(); ctx.errCnt=0;
ctx.KbCan = false; ctx.tm(TM_VL);
[Exit]
ctx.cans() should not be called immediately after the crc16 for a block has been written to the medium, or some CAN characters may be purged.
H
NON_CAN
[Entry]
[Exit]
onEvent(SER)
[(c==NAK || c=='C') & !ctx.KbCan]
/ctx.resendBlk();
ctx.errCnt++;
ctx.tm(TM_VL);
onEvent(SER)
[c==NAK & !ctx.KbCan]
/ctx.resendBlk(); ctx.errCnt++;
ctx.tm(TM_VL);
onEvent(SER)
[(c==ACK) & ctx.bytesRd & !ctx.KbCan]
/ctx.sendBlkPrepNext();
ctx.errCnt=0; ctx.tm(TM_VL);
ctx.firstBlk=false;
onEvent(SER)
[c==ACK &
ctx.fileName &
!ctx.KbCan]
/ctx.firstBlk= true;
ctx.tm(TM_VL);
onEvent(SER)
[c=='C' & ctx.bytesRd & !ctx.KbCan]
/ctx.sendBlkPrepNext();
ctx.tm(TM_VL); ctx.errCnt=0;
ACKNAKSTAT
[Entry]
[Exit]
One
[Entry]
[Exit]
ACKNAK
[Entry]
[Exit]
onEvent(SER)
[(c==NAK || (c=='C' & ctx.firstBlk)) &
(ctx.errCnt < errB) & !ctx.KbCan]
/ctx.resendBlk();
ctx.errCnt++; ctx.tm(TM_VL);
onEvent(SER)
[c==ACK & !ctx.fileName]
/ctx.result="Done";
onEvent(SER)
[c == 'C' & !ctx.bytesRd
& !ctx.KbCan]
/ctx.sendByte(EOT);
ctx.tm(TM_VL);
ctx.errCnt=0;
ctx.closeTransferredFile();
onEvent(SER)
[c=='C' & ctx.firstBlk &
ctx.errCnt < errB]
/ctx.sendByte(EOT);
++ctx.errCnt;
ctx.tm(TM_VL);
onEvent(SER)
[(c==ACK) & !ctx.bytesRd & !ctx.KbCan]
/ctx.sendByte(EOT);ctx.errCnt=0;
ctx.closeTransferredFile();
ctx.tm(TM_VL);
onEvent(SER)
[c=='C' & ctx.transferringFileD
!= -1]
/ctx.sendBlkPrepNext();
ctx.errCnt=0;
ctx.tm(TM_VL);
EOT1
[Entry]
[Exit]
onEvent(SER)
[c==NAK & !ctx.KbCan]
/ctx.sendByte(EOT);
ctx.errCnt=0;ctx.tm(TM_VL);
ctx.firstBlk=false;
EOTEOT
[Entry]
[Exit]
onEvent(SER)
[c == ACK & !ctx.KbCan]
/cout << "1st EOT ACK'd";
ctx.prepStatBlk(); ctx.tm(TM_VL);
onEvent(SER)
[c==ACK & !ctx.KbCan]
/ctx.prepStatBlk(); ctx.tm(TM_VL);
onEvent(SER)
[c==NAK &
!ctx.KbCan &
ctx.errCnt < errB]
/ctx.sendByte(EOT);
ctx.errCnt++;
ctx.tm(TM_VL);
StatC
[Entry]
[Exit]
onEvent(SER)
[c=='C' & ctx.fileName &
ctx.transferringFileD == -1]
/ctx.cans();
ctx.result="OpenError";
onEvent(KB_C)
/if (ctx.transferringFileD == -1) ctx.result="KbCancelledOpenErr";
else {ctx.closeTransferredFile(); ctx.result="KbCancelledFromStatC";}
onEvent(KB_C)
[!ctx.KbCan]
/ctx.KbCan = true;
ctx.tmRed(TM_VL -
TM_2CHAR);
onEvent(TM)
[!ctx.KbCan]
/ctx.tmPop();
onEvent(SER)
[c != CAN
& !ctx.KbCan]
/ctx.tmPop();
CAN
[Entry]
[Exit]
onEvent(KB_C)
/ctx.KbCan=true;
onEvent(SER)
[c == CAN]
/ctx.tmPush(
TM_CHAR);
onEvent(SER)
[c == CAN]
/ctx.closeTransferredFile();
ctx.clearCan();
ctx.result="RcvCancelled";
Unless a fast simulation has been chosen ...
TM_VL (Very Long timeout) gives 60 seconds
TM_SOH (normal timeout waiting for SOH) gives 10 seconds
TM_CHAR (inter-character timeout) gives 1 second
TM_2CHAR gives a period longer than the
inter-character timeout of 1 sec.
onEvent(SER)
[ctx.KbCan & (c==ACK || c==NAK ||
c=='C')]
/ctx.cans();
ctx.closeTransferredFile();
ctx.result="KbCancelled";
onEvent(SER)
[c==NAK & (ctx.errCnt >= errB)]
/ctx.cans();
ctx.closeTransferredFile();
ctx.result="ExcessiveNAKs";
Due to a bug in SmartState studio, in this StateChart, please interpret & as && -- a logical AND. Do not interpret it as a bitwise AND as you normally would.
YMODEM Sender Statechart
Copyright (c) 24 November 2021 Craig Scratchley
craig_scratchley AT alumni.sfu.ca
Real events are either
KB_C (cancel command received from keyboard, higher priority)
SER (serial character recieved, lower priority),
TM (timeout occurred)
onEvent(TM)
/ctx.cans();
ctx.closeTransferredFile();
if (ctx.KbCan)
ctx.result="KbCancelled";
else
ctx.result="Timeout";