MODULE *ReceiveModbus*

EXTENDS *Sequences*,
Naturals,
Modbus,
TLC,
ASCII,
FiniteSets

LOCAL INSTANCE *Hex*
WITH *natValue* ← 0, *hexValue* ← ⟨0⟩

LOCAL *Range*(*T*) ≜ {*T*[*x*] : *x* ∈ DOMAIN *T*}

MessagesToSerialPort ≜ these are in *ASCII* but they are converted to decimal before being used below. See *StrTupleToN*
{*StrTupleToNumTuple*(⟨":", "J", "G", "P", "9", "4", "3", "2", "J", "3", "9", "J", "G", "W", "I", "R", "W
StrTupleToNumTuple(⟨":", "\r", "\n"⟩),
StrTupleToNumTuple(⟨"J", "G", "P", "9", "4", "3", "2", "J", "3", "9", "J", "G", "W", "I", "R", "W", "
StrTupleToNumTuple(⟨":", "1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", "3", "7", "E", "\r", "\n
StrTupleToNumTuple(⟨":", "1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", "3", "7", "E", "C", "R"
StrTupleToNumTuple(⟨":", "1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", ":", "1", "1", "0", "3",
⟨⟩, ⟨1⟩, ⟨2⟩, ⟨3⟩, ⟨4⟩, ⟨5⟩, ⟨6⟩, ⟨7⟩, ⟨8⟩, ⟨9⟩, ⟨10⟩, ⟨11⟩, ⟨12⟩, ⟨13⟩, all possible values
⟨14⟩, ⟨15⟩, ⟨16⟩, ⟨17⟩, ⟨18⟩, ⟨19⟩, ⟨20⟩, ⟨21⟩, ⟨22⟩, ⟨23⟩, ⟨24⟩, ⟨25⟩, that could come across the serial line
⟨26⟩, ⟨27⟩, ⟨28⟩, ⟨29⟩, ⟨30⟩, ⟨31⟩, ⟨32⟩, ⟨33⟩, ⟨34⟩, ⟨35⟩, ⟨36⟩, ⟨37⟩, ⟨38⟩, ⟨39⟩, ⟨40⟩, ⟨41⟩,
⟨42⟩, ⟨43⟩, ⟨44⟩, ⟨45⟩, ⟨46⟩, ⟨47⟩, ⟨48⟩, ⟨49⟩, ⟨50⟩, ⟨51⟩, ⟨52⟩, ⟨53⟩, ⟨54⟩, ⟨55⟩, ⟨56⟩, ⟨57⟩,
⟨58⟩, ⟨59⟩, ⟨60⟩, ⟨61⟩, ⟨62⟩, ⟨63⟩, ⟨64⟩, ⟨65⟩, ⟨66⟩, ⟨67⟩, ⟨68⟩, ⟨69⟩, ⟨70⟩, ⟨71⟩, ⟨72⟩, ⟨73⟩,
⟨74⟩, ⟨75⟩, ⟨76⟩, ⟨77⟩, ⟨78⟩, ⟨79⟩, ⟨80⟩, ⟨81⟩, ⟨82⟩, ⟨83⟩, ⟨84⟩, ⟨85⟩, ⟨86⟩, ⟨87⟩, ⟨88⟩, ⟨89⟩,
⟨90⟩, ⟨91⟩, ⟨92⟩, ⟨93⟩, ⟨94⟩, ⟨95⟩, ⟨96⟩, ⟨97⟩, ⟨98⟩, ⟨99⟩, ⟨100⟩, ⟨101⟩, ⟨102⟩, ⟨103⟩, ⟨104⟩, ⟨105⟩,
⟨106⟩, ⟨107⟩, ⟨108⟩, ⟨109⟩, ⟨110⟩, ⟨111⟩, ⟨112⟩, ⟨113⟩, ⟨114⟩, ⟨115⟩, ⟨116⟩, ⟨117⟩, ⟨118⟩, ⟨119⟩, ⟨120⟩, ⟨121⟩,
⟨122⟩, ⟨123⟩, ⟨124⟩, ⟨125⟩, ⟨126⟩, ⟨127⟩, ⟨128⟩, ⟨129⟩, ⟨130⟩, ⟨131⟩, ⟨132⟩, ⟨133⟩, ⟨134⟩, ⟨135⟩, ⟨136⟩, ⟨137⟩,
⟨138⟩, ⟨139⟩, ⟨140⟩, ⟨141⟩, ⟨142⟩, ⟨143⟩, ⟨144⟩, ⟨145⟩, ⟨146⟩, ⟨147⟩, ⟨148⟩, ⟨149⟩, ⟨150⟩, ⟨151⟩, ⟨152⟩, ⟨153⟩,
⟨154⟩, ⟨155⟩, ⟨156⟩, ⟨157⟩, ⟨158⟩, ⟨159⟩, ⟨160⟩, ⟨161⟩, ⟨162⟩, ⟨163⟩, ⟨164⟩, ⟨165⟩, ⟨166⟩, ⟨167⟩, ⟨168⟩, ⟨169⟩,
⟨170⟩, ⟨171⟩, ⟨172⟩, ⟨173⟩, ⟨174⟩, ⟨175⟩, ⟨176⟩, ⟨177⟩, ⟨178⟩, ⟨179⟩, ⟨180⟩, ⟨181⟩, ⟨182⟩, ⟨183⟩, ⟨184⟩, ⟨185⟩,
⟨186⟩, ⟨187⟩, ⟨188⟩, ⟨189⟩, ⟨190⟩, ⟨191⟩, ⟨192⟩, ⟨193⟩, ⟨194⟩, ⟨195⟩, ⟨196⟩, ⟨197⟩, ⟨198⟩, ⟨199⟩, ⟨200⟩, ⟨201⟩,
⟨202⟩, ⟨203⟩, ⟨204⟩, ⟨205⟩, ⟨206⟩, ⟨207⟩, ⟨208⟩, ⟨209⟩, ⟨210⟩, ⟨211⟩, ⟨212⟩, ⟨213⟩, ⟨214⟩, ⟨215⟩, ⟨216⟩, ⟨217⟩,
⟨218⟩, ⟨219⟩, ⟨220⟩, ⟨221⟩, ⟨222⟩, ⟨223⟩, ⟨224⟩, ⟨225⟩, ⟨226⟩, ⟨227⟩, ⟨228⟩, ⟨229⟩, ⟨230⟩, ⟨231⟩, ⟨232⟩, ⟨233⟩,
⟨234⟩, ⟨235⟩, ⟨236⟩, ⟨237⟩, ⟨238⟩, ⟨239⟩, ⟨240⟩, ⟨241⟩, ⟨242⟩, ⟨243⟩, ⟨244⟩, ⟨245⟩, ⟨246⟩, ⟨247⟩, ⟨248⟩, ⟨249⟩,
⟨250⟩, ⟨251⟩, ⟨252⟩, ⟨253⟩, ⟨254⟩, ⟨255⟩
}

--fair algorithm *ReceiveModbus*
IPC calls
macro *send*(*dest*, *msg*) begin
sentBuffer := *sentBuffer* ∘ *msg*;

```
 end macro;

variables   rxBuf = ⟨⟩,
            rxReg = ⟨⟩,
            incomingMessages ∈ MessagesToSerialPort,
            incByte = ⟨⟩,
            msg = ⟨⟩,
            msgid = ⟨⟩,
            guid = ⟨0⟩,
            last2 = ⟨0, 0⟩,
            modchkBuffer = ⟨⟩,   this is what is passed to crypto. Only valid modbus here plz
            signBuffer = ⟨⟩      this is what is passed to modchk. Only valid modbus here plz

begin

trustnet_in1:  while Len(incomingMessages) > 0 do

                   ti1: msg := incomingMessages ;
                   ti2: incomingMessages := ⟨⟩ ;
               start:  while Len(msg) > 0 do                  while there are bytes left in the message

               inc:    if Len(msg) > 1 then                   pop off head of message
                           incByte := ⟨Head(msg)⟩ ;
                           msg := Tail(msg) ;
                       else
                           incByte := ⟨msg[1]⟩ ;
                           msg := ⟨⟩ ;
                       end if ;

                       rxReg := incByte ;

                       receive:    a ":" character indicates the start of a new message
                       if rxReg = StrTupleToNumTuple(⟨":"⟩)
                               then rxBuf := rxReg ;   restart the buffer essentially
                       end if ;

                        if the buffer is full then there is NO WAY it could be valid modbus
                       if Len(rxBuf) = MAXMODBUSSIZE then
           buffull:        rxBuf := ⟨⟩ ;
                           rxReg := ⟨⟩ ;
                           incByte := ⟨⟩ ;
                           last2 := ⟨0, 0⟩ ;
                           goto start ;
                       end if ;

                        only put character in buffer is there is already a ":" in it.
                        buffer can only start with ":" so if its empty then just discard character
```

```
            buffProp:    if Len(rxBuf) > 0 then
                             r0: last2  := Tail(last2 ∘ rxReg) ;   update last2
                             r1: rxBuf := rxBuf ∘ rxReg ;   put the contents of the register into the buffer
                         end if ;

                          empty the register
                         r2: rxReg := ⟨⟩ ;

            check:       if we get the end of the modbus "\r\n" then ship it
                         if NumTupleToStrTuple(last2) = ⟨"\r", "\n"⟩ then   convert back to ASCII before checkin
                             if (Len(rxBuf)) ≥ MINMODBUSSIZE then
                                 check0: msgid := ⟨guid[1]⟩ ∘ ⟨"t", "n", "i"⟩ ;
                                 check1: guid[1] := guid[1] + 1 ;
                                 check2: modchkBuffer := Append(modchkBuffer, [id ↦ msgid, text ↦ rxBuf,
                                 signBuffer := Append(signBuffer, [id ↦ msgid, text ↦ rxBuf]) ;
                                   check2: send("messagecheck", [id ↦ msgid, text ↦ rxBuf, source ↦ "trustnet_in"]);
                                        send("sign", [id ↦ msgid, text ↦ rxBuf]);
                             end if ;
                             check4:  rxBuf := ⟨⟩ ;
                             rxReg := ⟨⟩ ;
                             incByte := ⟨⟩ ;
                             last2 := ⟨0, 0⟩ ;
                         end if ;
                     end while ;
                 end while ;
end algorithm
```

BEGIN TRANSLATION

VARIABLES $rxBuf$, $rxReg$, $incomingMessages$, $incByte$, msg, $msgid$, $guid$, $last2$, $modchkBuffer$, $signBuffer$, pc

$vars \triangleq \langle rxBuf, rxReg, incomingMessages, incByte, msg, msgid, guid, last2, modchkBuffer, signBuffer, pc \rangle$

$Init \triangleq$ Global variables

$\wedge\ rxBuf = \langle\rangle$
$\wedge\ rxReg = \langle\rangle$
$\wedge\ incomingMessages \in MessagesToSerialPort$
$\wedge\ incByte = \langle\rangle$
$\wedge\ msg = \langle\rangle$
$\wedge\ msgid = \langle\rangle$
$\wedge\ guid = \langle 0 \rangle$
$\wedge\ last2 = \langle 0, 0 \rangle$
$\wedge\ modchkBuffer = \langle\rangle$
$\wedge\ signBuffer = \langle\rangle$
$\wedge\ pc =$ "trustnet_in1"

$trustnet\_in1 \triangleq \land pc = \text{“trustnet\_in1”}$
$\land \text{IF } Len(incomingMessages) > 0$
$\quad \text{THEN } \land pc' = \text{“ti1”}$
$\quad \text{ELSE } \land pc' = \text{“Done”}$
$\land \text{UNCHANGED } \langle rxBuf, rxReg, incomingMessages, incByte, msg, msgid, guid, last2, modchkBuffer, signBuffer \rangle$

$ti1 \triangleq \land pc = \text{“ti1”}$
$\land msg' = incomingMessages$
$\land pc' = \text{“ti2”}$
$\land \text{UNCHANGED } \langle rxBuf, rxReg, incomingMessages, incByte, msgid, guid, last2, modchkBuffer, signBuffer \rangle$

$ti2 \triangleq \land pc = \text{“ti2”}$
$\land incomingMessages' = \langle \rangle$
$\land pc' = \text{“start”}$
$\land \text{UNCHANGED } \langle rxBuf, rxReg, incByte, msg, msgid, guid, last2, modchkBuffer, signBuffer \rangle$

$start \triangleq \land pc = \text{“start”}$
$\land \text{IF } Len(msg) > 0$
$\quad \text{THEN } \land pc' = \text{“inc”}$
$\quad \text{ELSE } \land pc' = \text{“trustnet\_in1”}$
$\land \text{UNCHANGED } \langle rxBuf, rxReg, incomingMessages, incByte, msg, msgid, guid, last2, modchkBuffer, signBuffer \rangle$

$inc \triangleq \land pc = \text{“inc”}$
$\land \text{IF } Len(msg) > 1$
$\quad \text{THEN } \land incByte' = \langle Head(msg) \rangle$
$\qquad \land msg' = Tail(msg)$
$\quad \text{ELSE } \land incByte' = \langle msg[1] \rangle$
$\qquad \land msg' = \langle \rangle$
$\land rxReg' = incByte'$
$\land pc' = \text{“receive”}$
$\land \text{UNCHANGED } \langle rxBuf, incomingMessages, msgid, guid, last2, modchkBuffer, signBuffer \rangle$

$receive \triangleq \land pc = \text{“receive”}$
$\land \text{IF } rxReg = StrTupleToNumTuple(\langle \text{“:”} \rangle)$
$\quad \text{THEN } \land rxBuf' = rxReg$
$\quad \text{ELSE } \land \text{TRUE}$
$\qquad \land rxBuf' = rxBuf$
$\land \text{IF } Len(rxBuf') = MAXMODBUSSIZE$
$\quad \text{THEN } \land pc' = \text{“buffull”}$
$\quad \text{ELSE } \land pc' = \text{“buffProp”}$
$\land \text{UNCHANGED } \langle rxReg, incomingMessages, incByte, msg, msgid, guid,$

$last2,\ modchkBuffer,\ signBuffer\rangle$

$buffull \triangleq \land pc = \text{“buffull”}$
$\land rxBuf' = \langle\rangle$
$\land rxReg' = \langle\rangle$
$\land incByte' = \langle\rangle$
$\land last2' = \langle 0, 0\rangle$
$\land pc' = \text{“start”}$
$\land \text{UNCHANGED } \langle incomingMessages,\ msg,\ msgid,\ guid,\ modchkBuffer,$
$signBuffer\rangle$

$buffProp \triangleq \land pc = \text{“buffProp”}$
$\land \text{IF } Len(rxBuf) > 0$
$\text{THEN } \land pc' = \text{“r0”}$
$\text{ELSE } \land pc' = \text{“r2”}$
$\land \text{UNCHANGED } \langle rxBuf,\ rxReg,\ incomingMessages,\ incByte,\ msg,$
$msgid,\ guid,\ last2,\ modchkBuffer,\ signBuffer\rangle$

$r0 \triangleq \land pc = \text{“r0”}$
$\land last2' = Tail(last2 \circ rxReg)$
$\land pc' = \text{“r1”}$
$\land \text{UNCHANGED } \langle rxBuf,\ rxReg,\ incomingMessages,\ incByte,\ msg,\ msgid,$
$guid,\ modchkBuffer,\ signBuffer\rangle$

$r1 \triangleq \land pc = \text{“r1”}$
$\land rxBuf' = rxBuf \circ rxReg$
$\land pc' = \text{“r2”}$
$\land \text{UNCHANGED } \langle rxReg,\ incomingMessages,\ incByte,\ msg,\ msgid,\ guid,$
$last2,\ modchkBuffer,\ signBuffer\rangle$

$r2 \triangleq \land pc = \text{“r2”}$
$\land rxReg' = \langle\rangle$
$\land pc' = \text{“check”}$
$\land \text{UNCHANGED } \langle rxBuf,\ incomingMessages,\ incByte,\ msg,\ msgid,\ guid,$
$last2,\ modchkBuffer,\ signBuffer\rangle$

$check \triangleq \land pc = \text{“check”}$
$\land \text{IF } NumTupleToStrTuple(last2) = \langle \text{“\textbackslash r”},\ \text{“\textbackslash n”} \rangle$
$\text{THEN } \land \text{IF } (Len(rxBuf)) \geq MINMODBUSSIZE$
$\text{THEN } \land pc' = \text{“check0”}$
$\text{ELSE } \land pc' = \text{“check4”}$
$\text{ELSE } \land pc' = \text{“start”}$
$\land \text{UNCHANGED } \langle rxBuf,\ rxReg,\ incomingMessages,\ incByte,\ msg,\ msgid,$
$guid,\ last2,\ modchkBuffer,\ signBuffer\rangle$

$check4 \triangleq \land pc = \text{“check4”}$
$\land rxBuf' = \langle\rangle$

$\land\ rxReg' = \langle\rangle$
$\land\ incByte' = \langle\rangle$
$\land\ last2' = \langle 0, 0\rangle$
$\land\ pc' =$ "start"
$\land$ UNCHANGED $\langle incomingMessages, msg, msgid, guid, modchkBuffer, signBuffer\rangle$

$check0 \triangleq$ $\land\ pc =$ "check0"
$\land\ msgid' = \langle guid[1]\rangle \circ \langle$ "t", "n", "i" $\rangle$
$\land\ pc' =$ "check1"
$\land$ UNCHANGED $\langle rxBuf, rxReg, incomingMessages, incByte, msg, guid, last2, modchkBuffer, signBuffer\rangle$

$check1 \triangleq$ $\land\ pc =$ "check1"
$\land\ guid' = [guid$ EXCEPT $![1] = guid[1] + 1]$
$\land\ pc' =$ "check2"
$\land$ UNCHANGED $\langle rxBuf, rxReg, incomingMessages, incByte, msg, msgid, last2, modchkBuffer, signBuffer\rangle$

$check2 \triangleq$ $\land\ pc =$ "check2"
$\land\ modchkBuffer' = Append(modchkBuffer, [id \mapsto msgid, text \mapsto rxBuf, source \mapsto$ "trustnet_in"$])$
$\land\ signBuffer' = Append(signBuffer, [id \mapsto msgid, text \mapsto rxBuf])$
$\land\ pc' =$ "check4"
$\land$ UNCHANGED $\langle rxBuf, rxReg, incomingMessages, incByte, msg, msgid, guid, last2\rangle$

$Next \triangleq$ $trustnet\_in1 \lor ti1 \lor ti2 \lor start \lor inc \lor receive \lor buffull$
$\lor\ buffProp \lor r0 \lor r1 \lor r2 \lor check \lor check4 \lor check0 \lor check1$
$\lor\ check2$
$\lor$ Disjunct to prevent deadlock on termination
$(pc =$ "Done" $\land$ UNCHANGED $vars)$

$Spec \triangleq$ $\land\ Init \land \Box[Next]_{vars}$
$\land\ \mathrm{WF}_{vars}(Next)$

$Termination \triangleq \Diamond(pc =$ "Done"$)$

END TRANSLATION

receive buffer never overflows
$SAF1 \triangleq Len(rxBuf) \leq MAXMODBUSSIZE$
sending buffer never overflows
$SAF2 \triangleq$
$\land\ \forall\, x \in Range(signBuffer) : Len(x.text) < MAXMODBUSSIZE$
$\land\ \forall\, x \in Range(modchkBuffer) : Len(x.text) < MAXMODBUSSIZE$
$last2$ buffer always less than 3
$SAF3 \triangleq Len(last2) < 3$

only well-formed modbus gets forwarded

$$SAF4 \triangleq \\ \quad \wedge \forall x \in Range(signBuffer) : IsWellformedModbus(NumTupleToStrTuple(x.text)) \\ \quad \wedge \forall x \in Range(modchkBuffer) : IsWellformedModbus(NumTupleToStrTuple(x.text))$$

each message that is forwarded has a unique message *id*

$$SAF5 \triangleq \\ \quad \wedge \forall x \in Range(signBuffer) : Cardinality(\{y \in Range(signBuffer) : x.id = y.id\}) = 1 \\ \quad \wedge \forall x \in Range(modchkBuffer) : Cardinality(\{y \in Range(modchkBuffer) : x.id = y.id\}) = 1$$

well-formed messages get sent to both inner components

$$SAF6 \triangleq \\ \quad \wedge \forall x \in Range(signBuffer) : \exists y \in Range(modchkBuffer) : x.id = y.id \\ \quad \wedge \forall x \in Range(modchkBuffer) : \exists y \in Range(signBuffer) : x.id = y.id$$

rxBuf is either empty or starts with ":"

$$SAF7 \triangleq \neg(rxBuf = \langle\rangle) \Rightarrow Head(rxBuf) = CharToNum(\text{":"})$$

if the message is well-formed then it gets sent

$$LV1 \triangleq IsWellformedModbus(NumTupleToStrTuple(msg)) \rightsquigarrow \exists x \in Range(signBuffer) : x.text = msg$$ this nee

all messages are processed

$$LV2 \triangleq \Diamond\Box(incomingMessages = \langle\rangle)$$

*last*2 buffer gets reset after each well-formed message

$$LV3 \triangleq NumTupleToStrTuple(last2) = \langle \text{"\textsf{\textbackslash r}"}, \text{"\textsf{\textbackslash n}"} \rangle \rightsquigarrow last2 = \langle 0, 0 \rangle$$

* Modification History
* Last modified *Mon Jun* 03 22:20:42 *EDT* 2019 by *mssabr*01
* Last modified *Mon* May 14 12:52:02 *EDT* 2018 by *SabraouM*
* Created Sat May 05 11:36:54 *EDT* 2018 by *SabraouM*