Short range radio network (not WiFi). SLAVE module. Uses cheapie 433MHz modules or HC-12

433mhz_radio.zip

The following is a protocol I developed for a short range radio network without all the fluff associated with establishing and using a WiFi Ethernet presence. It is a simple star network with a master device initiating all communications to addressed slave modules. Messages may be passed back and forth- the content and interpretation of which is down to the application.

VB6 source code for the master is attached.

HC-12

A note on using cheap 433MHz modules.

The code as written used slaves with HC-12 modules. These are very nice and provide two-way communications in a single module. Cheap, 99cent 433MHz modules will only provide comms in one direction; as this software expects slaves to respond - at least as part of discovery - you will need to use Rx/Tx module pairs in the slave units which pushes up cost, PCB real-estate, circuity requirements (possibly two antennae for longer range applications) and software complications. The HC-12s, although a bit more expensive (beware of fake parts), do look very attractive with their simplicity and enhanced features from the onboard controller. Cheap modules operate in isolation, meaning the slave module would “hear” any transmissions from itself (this is mitigated by the HC-12s controller). The software should handle this echo (because the destination will not be the slave's address and so be ignored), but the incoming buffer will likely have content after a transmit. Be aware also that serial comms normally rests in a marking state - a logic 1 on the data (or ATAD :o) pin will leave the Tx module transmitting constantly - probably jamming your network quite effectively! The serial port will need closing after use and the Tx pin taken LO - this in itself complicates Rx as with the comms closed, you won't hear inbound messages… you'll probably have to mess about with splitting the comms port - but this would be an ideal application of the SerialTX CSUB. Rx needs a proper comm port (you need the buffer because of the arbitrary nature of data arrival). With all these wrinkles, my recommendation is stick with HC-12s, they are only $5 for the genuine article. In this case, the code as shown here should work “out of the box”.

With all this in mind, even at 99c each the cheap modules come with headaches - at least in this application. I would gladly pay the other $3 to be without them!

Protocol:

```
Master is address 000
Slaves are address 1-998
Broadcast is address 999

Master initiates all communication and slaves only ever respond to Master
requests for interaction (STAT or POLL).
Packets consist of a 4 character verb and then the payload.

All devices must be on the same "channel". It was explored that each slave
might exist on its own channel but this was abandoned
```

```
early on because the cons outweighed the pros:

Separate channel:   Pro    slave does not need to parse the addresses from
the packet-if it hears the packet then it must be for that slave.
                         Con    Broadcast becomes impossible.
                                Device addresses limited to 127.
                                Cheap RxTx modules have no concept of
channel.

Single channel:         Pro    Slave address is encoded as part of the
packet header with a maximum of 998 slave devices. Different channels
                               permit single master with multiple slave
zones if required, thus giving < 128000 devices (but broadcast and use of
                               cheapie modules will be impacted).
                               Broadcast is simple-all devices hear the
broadcast and pick the addresses from the packet as normal.
                               Can use cheapie 433MHz TxRx pairs with no
onboard processing (see preamble).
                         Con    HC-12 module is more expensive


Command     Source  Details
--------------+-------+-----------------------------------------
STAT:           Master  Master STATs address.
                Slave MUST respond within 1.5 seconds and may send back
data, thus a STAT contains an implicit POLL.
                Nil response will result in the slave marked as not
available in the status register (master will not POLL).
                Slave must reply with ACK0 only-If the Slave has more to
say, it must wait for a poll.
POLL:           Master  Master POLLs address.
                Slave may respond but must do so within within 1.5 seconds
if it has something to say but keeps quiet otherwise.
ACK0:           Slave   This is the terminating response to a STAT or POLL.
This packet will cause the Master to accept the response and
                move on to the next section of the cycle.
ACK1:           Slave   This is the response to a POLL only but signifies
the slave has more to report. This packet will cause the
                Master to accept the response and then immediately issue
another POLL to the same slave. Multiple ACK1's may be
                sent and the Master will continue to POLL and accept packets
until receipt of ACK0 or a timeout occurs, at which
                point the master will move on to the next section of the
cycle. The slave must pause for 1.5 seconds between each
                ACKx packet. An improvement would be to limit the amount of
ACK1s to prevent a slave blocking the master.
INIT:       Master    The addressed slave must restart.
SAFE:       Master    The addressed slave must go to its safe mode state.

Broadcasts: Any packet with destination address 999 is assumed to be a
```

```
broadcast. Usually only the Master Broadcasts. Slaves must not reply to
broadcasts.
TIME:       Master    The payload is hh:mm:ss,dd/mm/yyyy
                All slaves will respond to this packet and synchronize their
clocks to the payload.
INIT:       Master    All slaves must restart.
SAFE:       Master    All slaves must go to their safe mode state.
FIRE:       Master    All slaves must go to their FIRE mode state.


Slaves must respond to STAT. The response (or lack of it) updates the
internal status register for that slave. Slaves that do not
respond to a STAT will be skipped in the POLL cycle. The master will attempt
to discover all addresses continually.

Master: STAT (nn)
    Slave:    Must reply with ACK0 to the Master. The payload may be empty.
Failure to respond will record that address as
    unavailable/empty in the slave register.

Master: POLL (nn)
    Slave:    May reply with ACK0 (or ACK1 if there are multiple payloads)
to the Master. The payload may be empty but it is
    recommended the slave not respond to a POLL if it has nothing to report.
This reduces the cycle time and processing load.
    There is no advantage to replying with an empty payload.

The Packet format (all nodes):
Note the bracket forms below do not form part of the message and are here
simply to logically group sections.
[optional preamble]STX<checksum>GS{RC4<dest address><src
address><<Packet>}ETX

The data between the GS and ETX is encrypted before transmission. The
checksum is for all bytes between {}
RC4 (elsewhere on this wiki) is a complex algorithm with a moderate data
requirement (0.5K). A simpler encryption algorithm XORing a
variable length key is available if the simple nature of the slaves cannot
support it i.e. simple PIC/Atmel chips.
Smaller Micromites can struggle to maintain timings with RC4 and large
packets-use simple encryption or slow the cycle.

The payload must only contain "Printable" characters in the range 32-126.
This avoids erroneous control characters interfering with network operation.

A typical Tx from the master to device 05 looks like

Preamble: When using cheap TxRx modules with no onboard processing, it is
necessary to send a stream of 1's with occasional zero so the Rx
    can set it's levels before the actual data arrives.
    Without preamble, some bits may be lost while the Rx adjusts its levels.
```

```
Anything to the left of the STX must be discarded.
    Chr FE has a large amount of "1" space and the trailing 0 ensures the
levels are set on Rx devices
    immediately before they are required to be accurate. A string of at
least 2 FE chars is effective.
    Using devices like HC-12 where there is onboard processing to assure
data integrity, a preamble is not necessary as it is
    taken care of by the radio modem.

    [FE FE ...]

Packet structure
    02 = STX Start of Text, sync marker. This indicate the start of valid
data.
    CHK_HI ascii Hex of checksum of packet between the following GS and ETX
    CHK_LO
    1D = GS group separator, start of the addressing and instruction
details.
    30 = attention device 005 in ascii
    30
    35
    30 = from the master 000 in ascii
    30
    30
    50 Command verb, STAT, POLL, ACK0 etc...
    4F
    4C
    4C
    [ optional payload here ]
    03 = ETX End of Text, marks the end of the packet. No further data is
permitted.

Any response must begin transmission in the next 1.5 seconds
```

Master module.

You need to add your own code to interpret the content of the packets (both master and slaves) but standardise functionality so the master and slaves know what each other is doing. The master does not have to be a MicroMite-I have working VB code which uses a HC-12 on a USB↔Serial adapter so a Windows server can control the network-this opens up all sorts of possibilities and is used in the Snooker Hall metering for club membership and time charging etc.

Slaves and (micromite)master log to their console but this may be disconnected in general use. The modules detect “first run” and will prompt for config (i.e. their address and name-which is usually the location) at the console, after that it stores it's config and just gets on with things. So use the console to set the thing up and then just leave it to run. The modules I built have their console connections brought to a 3.5mm stereo jack which goes to a USB↔Serial jobby plugged in the installers laptop, just plug in to each and turn on, configure and run then unplug and move on to the next.

The master STATs every address from 001 to 998 every so often and if a module responds, it is marked as “live” and recorded along with it's name. Live Slave modules are POLLed in order for any messages interspersed with the next discovery. So the cycle might look like:

```
Stat 1 (responds)
Poll 1
Stat 2
Poll 1
Stat 3 (responds)
Poll 3
Stat 4
Poll 1
Stat 5
Poll 3
Stat 6
Poll 1
...
Stat 998
Time
```

Only known live salves are regularly polled and newly discovered slaves are added as they are discovered. This ensures that we discover all modules over time but do not delay polling known live modules.

Undiscovered addresses are polled continually in the above cycle allowing “live” addition of new modules without the need to re-initialize the master.

At the end of the cycle, a TIME packet is sent on the broadcast address and all slaves set their clocks.

INIT message can be sent to restart an individual module or broadcast to restart the entire network. SAFE message can be sent to individual or all modules to set them to their safe mode (maybe an engineer is about to open them up and doesn't want a mains circuit up his arm). FIRE message can be broadcast to set all modules in their default emergency mode - perhaps turn all the lights on, play a recorded message to evacuate the building etc…

Slave module.

You will need to adapt it as you see fit but it works really well and offers some nice features.

Slave modules listen to radio transmissions for their address or the broadcast address and respond accordingly. I have used this for a number of different projects: Snooker/Pool table light control, Customer feedback pods etc.

The Master initiates all communication and transmits structured and encrypted packets for specific slave addresses.

The Master is always address 000 and slaves can be any from 001 to 998 (doesn't have to be contiguous). 999 is the broadcast address. You could easily extend this but probably a thousand slave modules will cover most installations. You don't have to use them all and the master won't waste time trying to talk to addresses that don't respond to discovery.

The master also sends out TIME packets so slaves don't need RTCs (much cheapness!), only a single RTC on a network of 1000 modules!

Slave Module Code:

```
    Option Autorun On

    Print "Node boot"
    timer=0

    Option Base 0
    'Option Explicit

    Const
KEY$=">4!1x4q3z4+7%4{9?5\3HhH^5$9=6@1~6,7_7|1)7'3]7[9:8<3*8S9I9l7Z1eT0r1"
    Const GS=&H1D
    Const STX=2
    Const ETX=3
    Const io=1
    Const DoPreamble=0    ' No. of preamble bytes. Only for when using cheap
dumb RxTx module pairs. Drop the baud rate right down (2400 ish) And set
this value to 2 or more
    Const Link$="COM1:9600,260"

    Dim MyAddr As Integer
    Dim MyJob As String

    Dim a$, tm$, cmd$, pl$, R ' R is the serial buffer
    Dim Integer c,src,dst,n
    Dim Integer FLAGS=0  ' 0 STX
                         ' 1 GS
                         ' 2 ETX
                         ' 3 Init. device has restarted And never heard
from the master
                         ' 4 in FIRE mode
                         ' 5 in SAFE mode
                         ' 6 just been configured
                         ' 7
                         ' 8
                         ' 9
                         '10
                         '11
                         '12
                         '13
                         '14
                         '15
                         '16
                         '17
                         '18
                         '19
```

```
                              '20
                              '21
                              '22
                              '23
                              '24
                              '25
                              '26
                              '27
                              '28
                              '29
                              '30
                              '31

    Var Restore

    FlagSet 3 ' just booted

    IF MyAddr=0 Then
        Do
            Do
                Input "Enter Node address (1-998) >", a$
                MyAddr=int(val("0"+a$))
            Loop Until MyAddr >0 And MyAddr<999

            Do
                Input "Enter Node Task Name       >", a$
                Myjob=A$
            Loop Until MyJob<>""

            Print "Is This Correct? (Y/N)";
            Do:Input a$:a$=UCASE$(a$):Loop Until a$="Y" OR a$="N"
        Loop While a$="N"

        Var Save MyAddr,MyJob

        FlagSet 6 ' Just completed setup-need INIT
    EndIf

    Open Link$ As #io ' may need to drop this for cheap TxRx modules, HC12
etc do not require
    FlushIO io

    Print "Node";MyAddr;" up in";Timer;"mS Name ";MyJob

    Do
        If Loc(#io)<>0 Then
            C=Asc(Input$(1,#io))
            Select Case C
                Case 2
                    R=""
                    If FLAGS<>0 Then Goto SafeExit
```

```
                    FlagSet 0
                Case &H1D    ' test GS
                    If Flagtest(0)<>1 And FlagTest(2)<>0 Then Goto SafeExit
                    FlagSet 1
                    R=R+","
                Case 3
                    If Flagtest(0)<>1 And FlagTest(2)<>1 Then Goto SafeExit
                    FlagSet 2
                Case &H30 TO &h39
                    If Flagtest(0)<>1 Then Goto SafeExit ' only accept chars
after STX
                    R=R+Chr$(C)
                Case &h41 TO &H46
                    If Flagtest(0)<>1 Then Goto SafeExit
                    R=R+Chr$(C)
                Case Else
                    'ignore anything else
            End Select
        EndIf

        If (FLAGS And 7)=7 Then    'STX,GS,ETX


            ' check GS position(5)
            c=Instr(R,",")
            If c<>5 Then Goto BadFrame '? "GS bad position":

            ' data after GS should always be even count
            c=Len(Mid$(R,6))
            If (c And 1) Then Goto BadFrame '? "Data length is odd":GOTO
BadFrame ' If bit0 set then odd number

            'calculate checksum of rec'd data
            c=Val("&H"+Left$(r,4))
            a$=Mid$(R,6) 'payload

            For n=1 To Len(a$):c=c-Asc(Mid$(a$,n,1)):Next

            If c<>0 Then Goto BadFrame '? "bad checksum ";r:GOTO BadFrame
            ' For some reason, slaves calculate bad checksums For each
others reply packets.
            ' haven't determined why but it gives a quick exit If the packet
isn't For me

            'decode valid frame

            Timer=0
            a$=DECRYPT$(a$)
dst=Val(Left$(a$,3)):src=Val(Mid$(a$,4,3)):cmd$=Mid$(a$,7,4):pl$=Mid$(a$,11)
```

```
            If dst=MyAddr Then ' specific
                Print Str$(dst,3);" ";TIME$;" ";cmd$;" ";pl$';Timer

                Select Case cmd$
                    Case "STAT" ' must reply with info
                        RadioSend
src,MyAddr,"ACK0:STAT"+Bin$(FLAGS,32)+":NAME="+MyJob+":NTYP="+MM.Device$+","
+Str$(MM.Ver)
                    Case "POLL" ' may reply with data
                        If FlagTest 6=0 Then 'If 1 we are waiting INIT so do
not answer
                            If int(Rnd*2))>0 Then ' decide To answer or not
                                RadioSend src,MyAddr,"ACK0"
                            EndIf
                        EndIf
                    Case "INIT"
                        CPU Restart
                    Case "SAFE"
                        Mode 5,pl$
                    ' ignore anything else
                End Select

            ElseIf dst=999 Then ' broadcast only
                Print Str$(dst,3);" ";Time$;" ";cmd$;" ";pl$
                Select Case cmd$
                    Case "TIME"
                        If Len(pl$)=19 Then
                            If Mid$(pl$,9,1)="," Then
                                On Error Skip 1:Time$=Left$(pl$,8)
                                On Error Skip 1:Date$=Right$(pl$,10)
                            EndIf
                        EndIf
                    Case "INIT"
                        CPU Restart
                    Case "SAFE"    ' safe mode For working on node
                        Mode 5,pl$
                    Case "FIRE" ' emergency mode-i.e. all nodes turn on
lights
                        Mode 4,pl$
                    ' other broadcast CASEs go here
                End Select
            EndIf

            Goto SafeExit
        EndIf

        Goto LoopEnd
BadFrame:
    ' ignore it
SafeExit:
    FlushIO(io)
```

```
LoopEnd:
    Loop

    Sub Mode(fg As Integer,x$)
        If x$="OFF" Then
            FlagRes fg
        ELSE
            FlagSet fg
        EndIf
    End Sub

    Function ZPad$(x As Integer)
        ZPad$=Right$("000"+Str$(x), 3)
    End Function

    Sub RadioSend(d As Integer,s As Integer,msg As String)
        Local qq$
        Local Integer chk,m
        qq$=ZPad$(d)+ZPad$(s)+msg
        For m=1 To Len(qq$):chk=chk+Asc(Mid$(qq$,m,1)):Next
        qq$=Chr$(STX)+HEX$(chk,4)+Chr$(GS)+ENCRYPT$(qq$)+Chr$(ETX)
        Print#io, String$(DoPreamble,Chr$(&HFE))+qq$;
        Pause Len(qq$)' wait For the packet To go
    End Sub

    Function ENCRYPT$(A$)
        Local Integer N,P
        Local B$
        P=1
        For N=1 To Len(A$)
            B$=B$+HEX$(Asc(Mid$(A$,N,1),2) Xor Asc(Mid$(KEY$,P,1)),2)
            P=P+1:If P>Len(KEY$) Then P=1
        Next
        ENCRYPT$=B$
    End Function

    Function DECRYPT$(A$)
        Local Integer N,P
        Local B$
        P=1
        For N=1 To Len(A$) Step 2
            B$=B$+Chr$(Val("&H"+Mid$(A$,N,2)) Xor Asc(Mid$(KEY$,P,1)))
            P=P+1:If P>Len(KEY$) Then P=1
        Next
        DECRYPT$=B$
    End Function

    Sub FlagSet(bit As Integer)
        FLAGS=FLAGS Or (2^bit)
    End Sub
```

```
    Sub FlagRes(bit As Integer)
        FLAGS=(FLAGS Or (2^bit)) Xor (2^bit)
    End Sub

    Function FlagTest(bit As Integer) As Integer
        FlagTest=Abs(Sgn(FLAGS And (2^bit)))
    End Function

    Sub FlushIO(CH)
        Local A$
        Do While LOC(#CH)>0
            A$=Input$(Min(Loc(#CH),255),#CH)
            Pause 20
        Loop
        R="":FLAGS=FLAGS And &HFFFFFFFFFFFFFFF8
    End Sub
```