- **DHT11 DHT22 Temperature and Humidity Sensor[ESP32FORTH]**
- **ORIGINAL ARTICLE : https://ohiyooo2.pixnet.net/blog/post/406078286**
- **BY Frank Lin 2022.7.20** Jul 23 Sat 2022 21:20

foreword

Nine times out of ten, the first temperature and humidity sensor that you will come into contact with when playing with Arduino is probably the DHT series sensor.

This series of sensors is really super simple to use because a bunch of people have written the underlying communication functions for it. It's no exaggeration, it's done in five minutes. So the author also used the same method and quickly developed a dew point temperature and humidity meter .

However, playing a language like ESP32FORTH, which is played by a small number of people, is naturally not so good. This kind of special agreement, but no one will care about you. Although ESP32FORTH is developed in C language, it is not difficult to "stick" the underlying communication functions of the DHT series developed by others in C language into the FORTH system.

However, in fact, it is not very difficult to build the code from scratch to communicate with the very special 1-wire communication protocol of the DHT series. Let's practice and enjoy implementing some special communication protocols according to the specification. Treat him as an after-dinner pastime, just like playing Sudoku! 😬

So let's use ESP32FORTH to implement the communication and reading of the two DHT series temperature and humidity dual sensors DHT11, DHT22.

DHT11 Temperature and Humidity Sensor

About DHT11, straight to the point, everything starts from this specification !

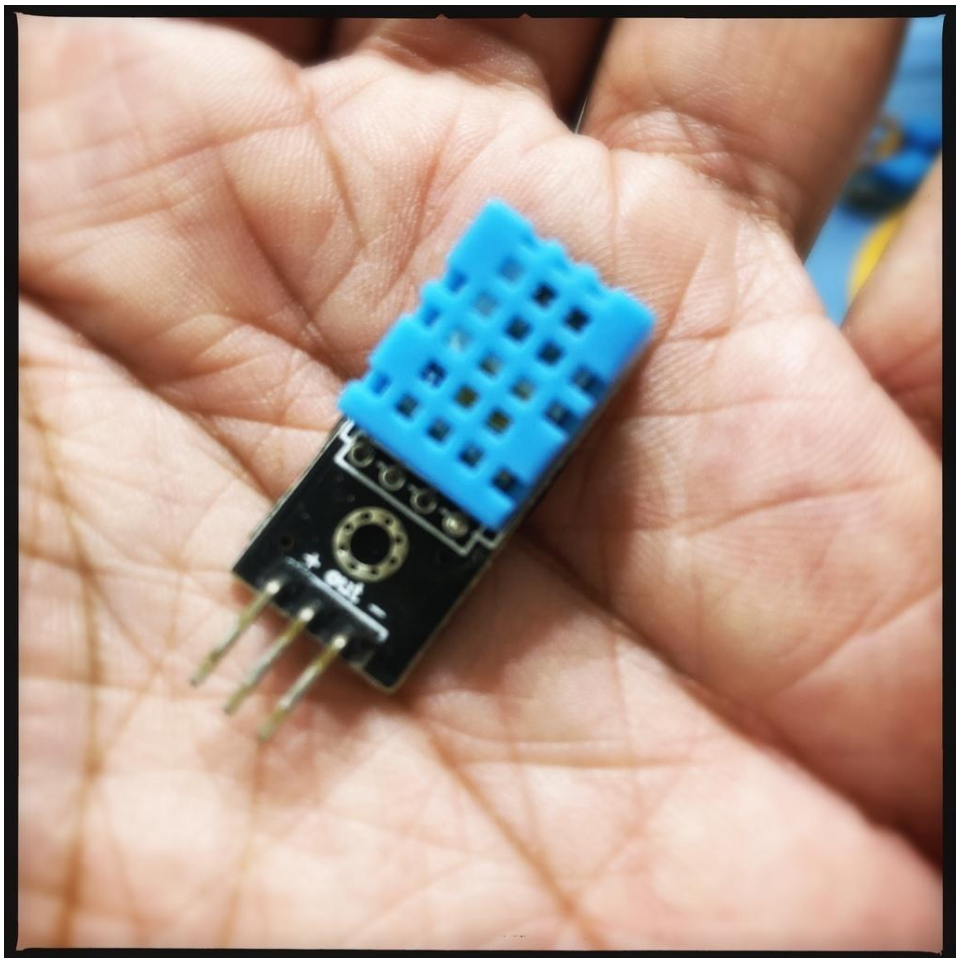
DHT series, this sensor used to be called Smart I/O, because it has built-in MCU itself, so we don't need to worry about the control of the analog temperature and humidity sensor. As long as they are properly triggered, the values of temperature and humidity are reported to our program in "digital" form through their special 1-wire interface.

So as long as you follow the specification, make the 1-wire communication protocol of the hardware layer into practice, and then decode the data according to the decoding format of the data, so it is completely OK!

DHT11 and DHT22 are the same series of temperature and humidity sensors. The DHT11 is a cheap version of the DHT22, and the specifications and accuracy are quite different from the DHT22. Personal experience, it is highly not recommended to buy DHT11, because it is really inaccurate. In order to make a dew point temperature hygrometer, I bought 4 or 5 DHT11s for cheap at first. But I was surprised to find

that there are so many pieces, and the reading value of each piece is very different. I really don't know which one is correct. Then there is the reading of humidity, which is more than 40% different from the hygrometer at home. It really makes people lose confidence, and DHT11 has since been included in the list of rejected households. After switching to DHT22, all accuracy problems were solved. Sure enough, you get what you pay for, expensive, and a guarantee of quality.

DHT11 Temperature Sensor

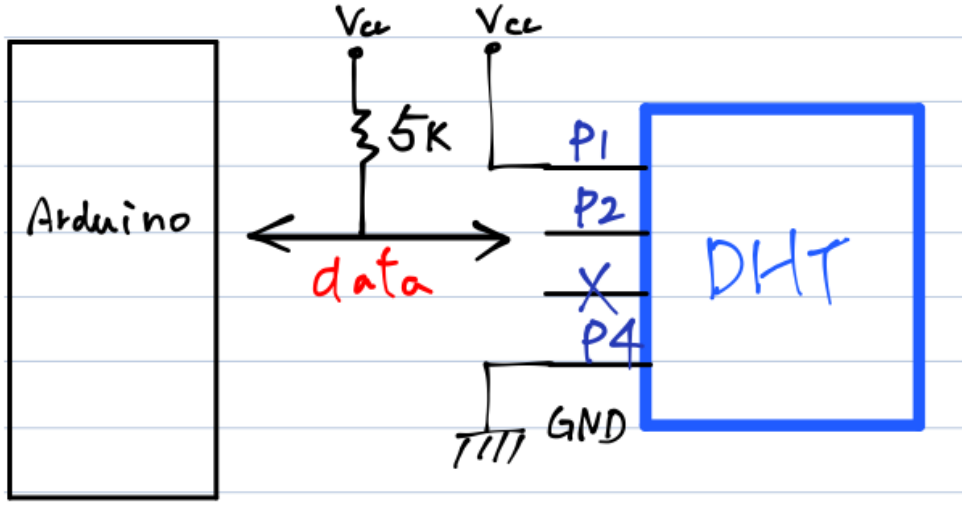


Let's first look at the specifications of DHT11

	Measurement range	Accuracy	Numerical resolution	Reaction time
temperature	0 - 50C	+/- 2C	1C	6s - 30s
humidity	20 - 90% RH	+/- 5%RH	1%	6s - 15s

Let's just say, the accuracy of +/-2C and +/-5%RH is a terrible specification! (Similar to a toy...) 😅

Hard wiring, the special 1-wire connection of DHT series is shown below



There are three main pins, two pins for power and ground. The other one is the Data pin. ESP32 uses digital I/O (DIO) to connect to this data pin, and a 5K pull-up resistor must be connected at the same time.

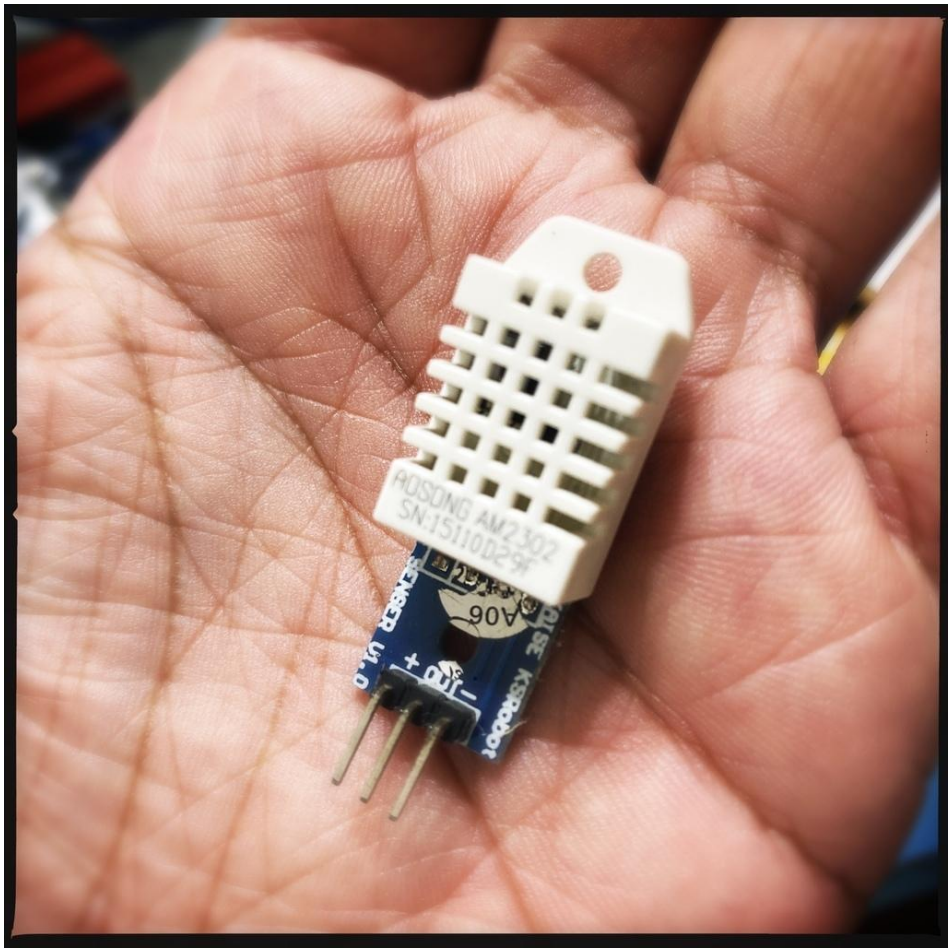
This kind of 1-wire trick is almost the same after watching it for a long time. All use pull-up resistors to achieve bidirectional, multiple Slaves communicate with the Master together. The key is also very simple. Usually, the DIO is in a listening state. At this time, the DIO must be high impedance. As a result of the pull-up resistor, as long as all the people on the line are in a high-impedance listening state, the natural potential is 3.3V (HIGH) level.

When someone changes from a high-impedance listening state to an output state of an output signal, when this DIO - HIGH (3.3V), everyone will still see HIGH (3.3V). But when this DIO is pulled LOW (0V), current flows and the pull-up resistor limits the current. At this time, everyone sees LOW. (Everyone else is still in high impedance listening state)So as long as only one DIO output is always turned on, everyone else maintains a high-impedance listening state, so that the multi-party conversation can continue.

Therefore, as long as one line is required, multiple people can exchange messages. The communication of information at the hardware layer will be explained later with the code, so that it will be clearer.

DHT22 Temperature and Humidity Sensor

The detailed specification is here , DHT22 is also known as AM2302.



Check out the specifications of DHT22

	Measurement range	repeatability	Numerical Analytical Accuracy	Reaction time
temperature	-40 - 80C	+/- 0.2C	0.1C	2s
humidity	20 - 90% RH	+/- 1% RH	0.1%RH	2s

It can measure to -40C, and the resolution, accuracy, and repeatability are also good. The author's actual experience after using it is very satisfactory! (However, the price is not cheap! 😅)

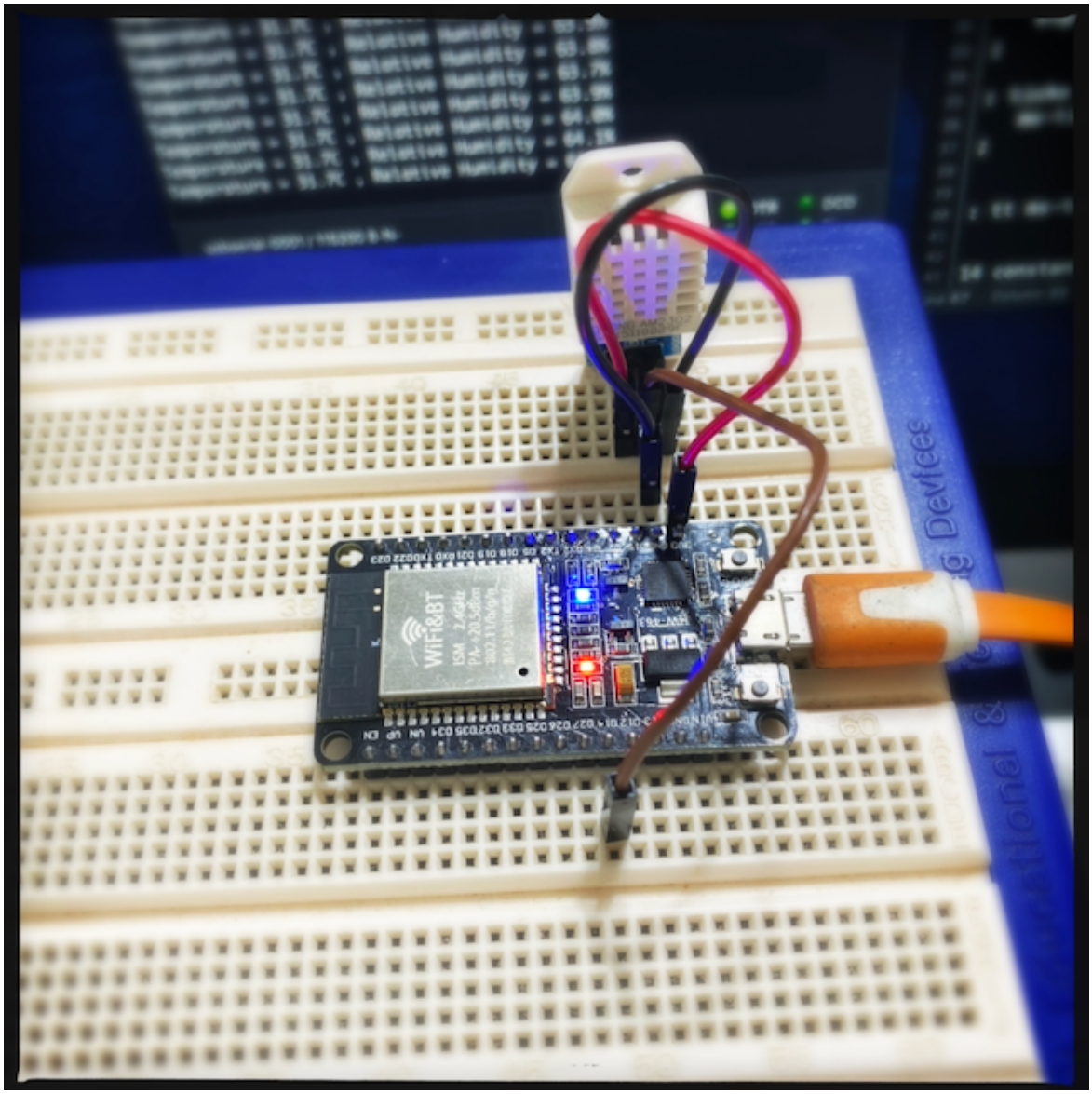
The wiring of DHT11 and DHT22 are the same. The communication format of the hardware layer is also the same. The only difference is in the format of the data encoding. In the same way, it will be clearer if the details are followed by the explanation with the code.

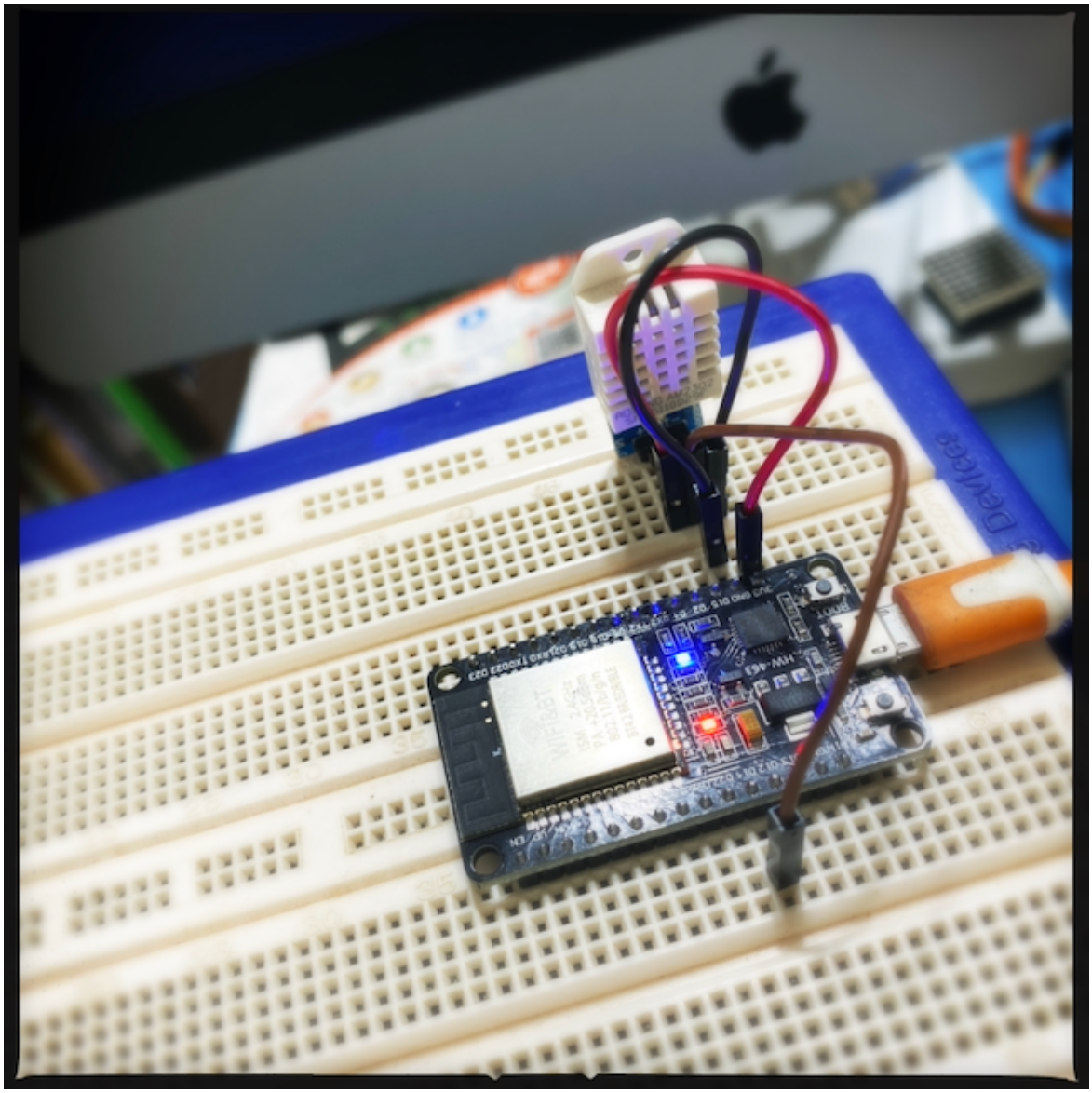
Hardwired

Because the module you bought is already a module, so even the pull-up resistor will help you get it done. Just simple wiring is fine.

Here it is very simple to connect Pin14 of ESP32 to Data Pin of DHT.

Connect Vcc/GND one by one, and you're done. It's super easy!

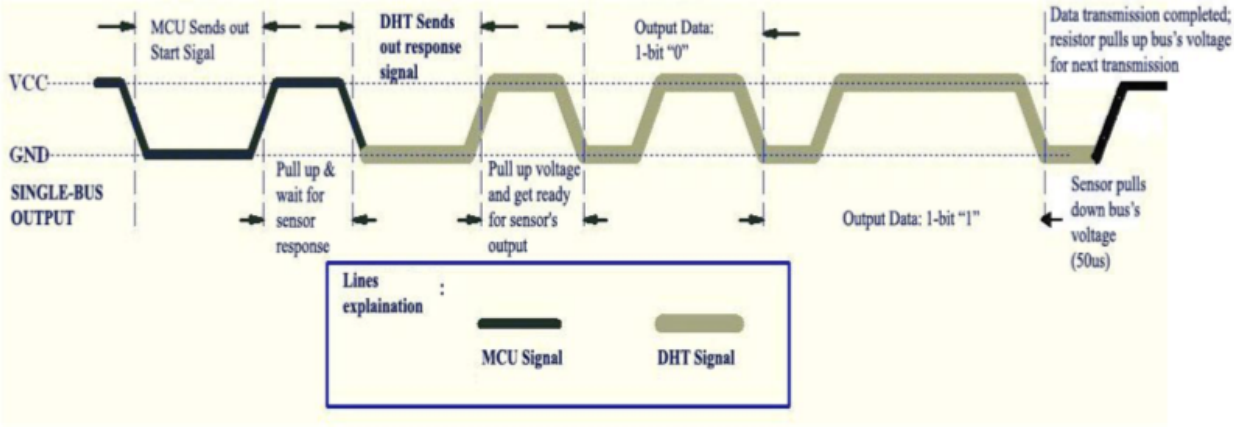




FORTH Code Explanation

DHT's 1-wire hardware layer protocol

As shown below, divided into two parts



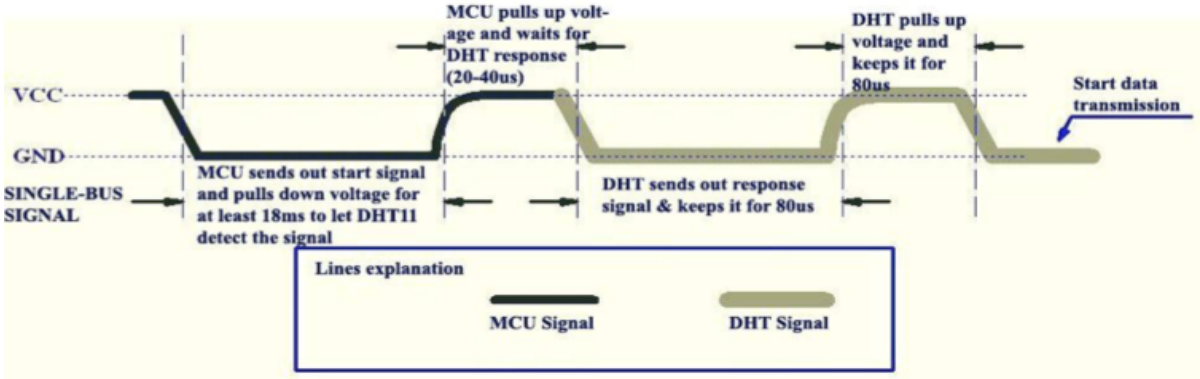
(1) MCU Start signal:

First, the MCU switches the DIO to the output state, pulls the signal line LOW to create a Start signal, and asks the DHT device that is listening, please send me the information. After transmitting the Start signal, the MCU immediately switches the DIO back to the high-impedance listening state to avoid interfering with the subsequent signal transmission of others on the signal line.

(2) DHT transmits data:

The DHT listens to the Start signal, confirms that the signal meets the requirements and accepts it, and immediately changes the DIO from the listening high impedance state to the output state. Through the good signal rules of the protocol, pull the signal line High and pull Low in sequence to transmit the subsequent handshake confirmation and 40 bits (5 bytes) data. The MCU listens to the signals of the signal lines, and sequentially reorganizes these signals back to the humidity and temperature information according to the agreed rules

.**(1) Start signal of MCU** As shown below,



It is very simple, when the MCU turns on the DIO output, and continuously pulls the signal line LOW (0V) for at least 18 mS (milliseconds), when the DHT hears such a signal, it will accept the trigger and is ready to continue to change the DIO from the high-impedance listening state. into the output state of voltage, and start outputting data.

After sending the Start signal, the MCU must put the DIO back to HIGH (5V) for about 20 - 40 uS (microseconds), and then immediately switch the DIO to a high-impedance listening state. Start listening for messages sent by the DHT.

This code is as follows, first define a delay command to control the time.

```
: delay for next ;
```

This instruction uses a loop to test (instruction t2), the time it takes to delay a delay will be 96.5ms / 1000000 = 0.0965 uS

Create a Start signal

```
: start! ( --)

   DHTPin >OUTPUT

   DHTPin ->Low

     20 ms

   DHTPin ->High

     145 delay ( ~ 14uS)

   DHTPin <INPUT


;
```

Look at the picture to tell the story, first cut the data DIO pin of DHTPin into OUTPUT state (>OUTPUT). Then pull Low, pull it High after 20 mS, and then give it a 14uS delay (145 delay), switch DIO back to the high-impedance listening state (<INPUT), and prepare to listen to the message from the DHT.

(2) DHT transmits data:

Preamble signal before DHT transmits data

In order to be afraid of problems, DHT will not send data immediately, but will first send the preamble signal as shown in the figure above to shake hands. The MCU uses this signal (code) to confirm whether it is talking to an alien or a real DHT. If it is a genuine DHT, it will pull the signal line to Low for about 80uS and then to High for 80uS. If this is not the case, then it is definitely not the DHT himself who is sending the message to you now! 🤦‍♀️

First define a listening command that waits for the signal line to pull High

```
: wait ( --) \ wait until pulse-high

   begin DHTPin Pin@ until ;
```

keep listening to DHTPin until the signal is not zero (Low)

Then there is the complete code that listens to all the data sent by the DHT

```
: DHT@ ( -- n1 n2 n3 n4 CheckSum)

   start!

   207 delay ( ~ 20uS)

   wait

   850 delay ( ~ 82uS) 40bits

   @

;
```

Send the Start signal first, and then we wait for 20uS (207 delay) to make sure that the DHT has entered the 80uS preamble signal Low, and then use wait to wait for the preamble signal to enter the 80uS High. Precisely wait for 82uS (850 delay) to let the signal enter the data transmission stage of DHT transmission of 40bits, and then hand it over to the 40bits@ relay to read 5 bytes (40bits) of humidity and temperature data one by one.

After the DHT sends the preamble signal, the next step is to send 5 bytes of data. 5 bytes of data has a total of 40 bits. The data of these five bytes, the first two bytes are humidity data, and the next two bytes

are temperature data. The last byte is the CheckSum check code, which is used to confirm whether the data of these 4 bytes are wrong. But it should be noted that although the hardware layer also transmits 5 bytes, the humidity and temperature data encoding of DHT11 and DHT22 are not the same. Therefore, after receiving the data, different encoding methods are used to restore the data.

Each byte of data has a total of 8 bits, and the message format sent by the hardware layer is as follows

Because the data line has only one line, the length of the pulse wave High is used to represent the 0 and 1 bit signal. A short pulse is 0, and a long pulse is 1.

Short pulse wave, indicating 0

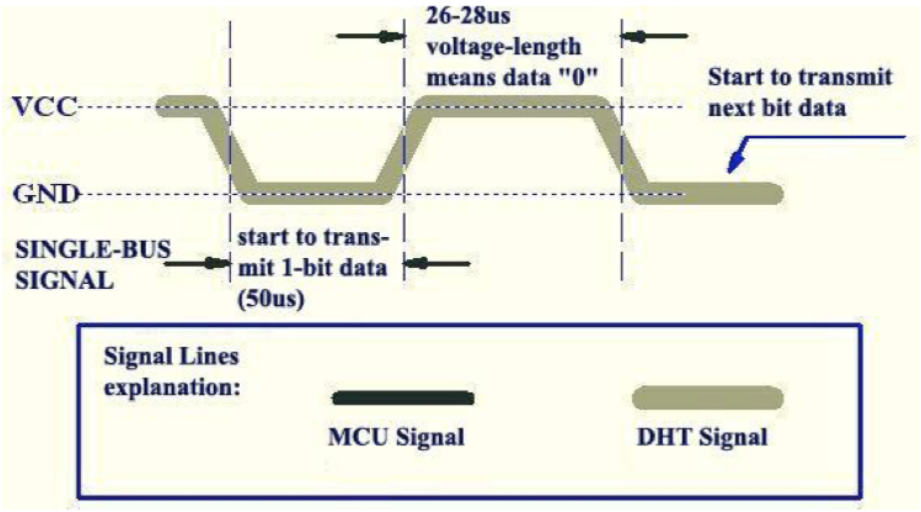


Figure 4 Data "0" Indication

First, the Start signal with a Low length of 50 uS will be pulled to inform and synchronize the data bit to be transmitted. **When the connected High signal is a short signal of 26 - 28 uS, it means this bit is 0**

Long pulse wave, indicating 1

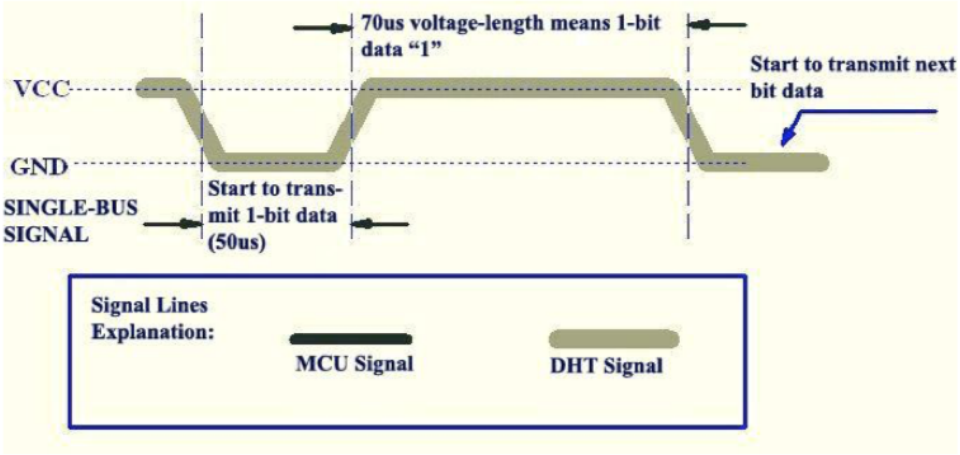


Figure 5 Data "1" Indication

First, the Start signal with a Low length of 50 uS will be pulled to inform and synchronize the data bit to be transmitted. **When the connected High signal is a 70 uS long signal, it means this bit is 1**

Therefore, to determine the code with a short pulse length of a single bit

```
: signal@ ( -- true=1/false=0)
   174 ( ~112uS)
   for
       DHTPin Pin@ 0= ( pulse low?)
       if r> 104 ( ~ 67.175uS) < exit ( length > 70 = 44.825 uS)
       then
   next
   ." Error! Signal not match with expectation!" cr
   abort
;
```

Use 175 loops to judge and measure the length of the High signal.

It should be noted that each execution time of this loop has been measured (using the t1 instruction), and each time it will take 642ms / 1000000 = 0.642 uS.

So 175 times will run at least 112 uS (=175*0.642 uS) far more than the long signal of 70uS. Therefore, when the loop is completed, the Low signal has not been seen yet. This must be a problem, it displays an error message that the signal is not as expected and stops execution (Abort).

In the loop, DHTPin Pin@ 0= is used to judge whether the signal has become Low. If it is Low, it is judged whether it is 0 or 1 according to the length.

The key is the count of the loop, r> will retrieve the current number of the loop. for-next is reciprocal. So when the number of this loop is less than 104, it means that the loop has been executed at least 175 - 104 = 71 times. The execution time of 71 times is the short pulse time of 71*0.642 = 45.582 uS > 28 uS.

Therefore, when it is found that the received pulse wave has changed from High to Low, at this time, if r> 104 < is established, it is a long pulse wave, representing 1; otherwise, it is a short pulse wave, representing 0. After getting the final result of the long and short pulse bit signal, exit jumps out of the instruction execution.

The entire 8 bits (1 byte) data read

```
: 8bits@ ( -- Data)

   0 ( data)

   7 for

        wait

       signal@ if 1 r@ lshift or then

     next

;
```

Use 8 for-next loops. Each time, use wait to wait for the end of the leading signal Low, and hand it over to the signal@ at the moment of turning to High to judge the length of the continuous High pulse wave, and see if it is 0 or 1.

The DHT data has instructions that they are sent high-order bits first. It happens that our for-next is reciprocal, so the sequential count is 7, 6, 5 ... 0. So when it is judged that it is 1, move 1 to the left by the number of digits counted by for-next at that time, and use or to merge the data of this element into data. Complete 8 times in sequence, and the data of this byte will be solved.

Because there are 5 bytes in total , so

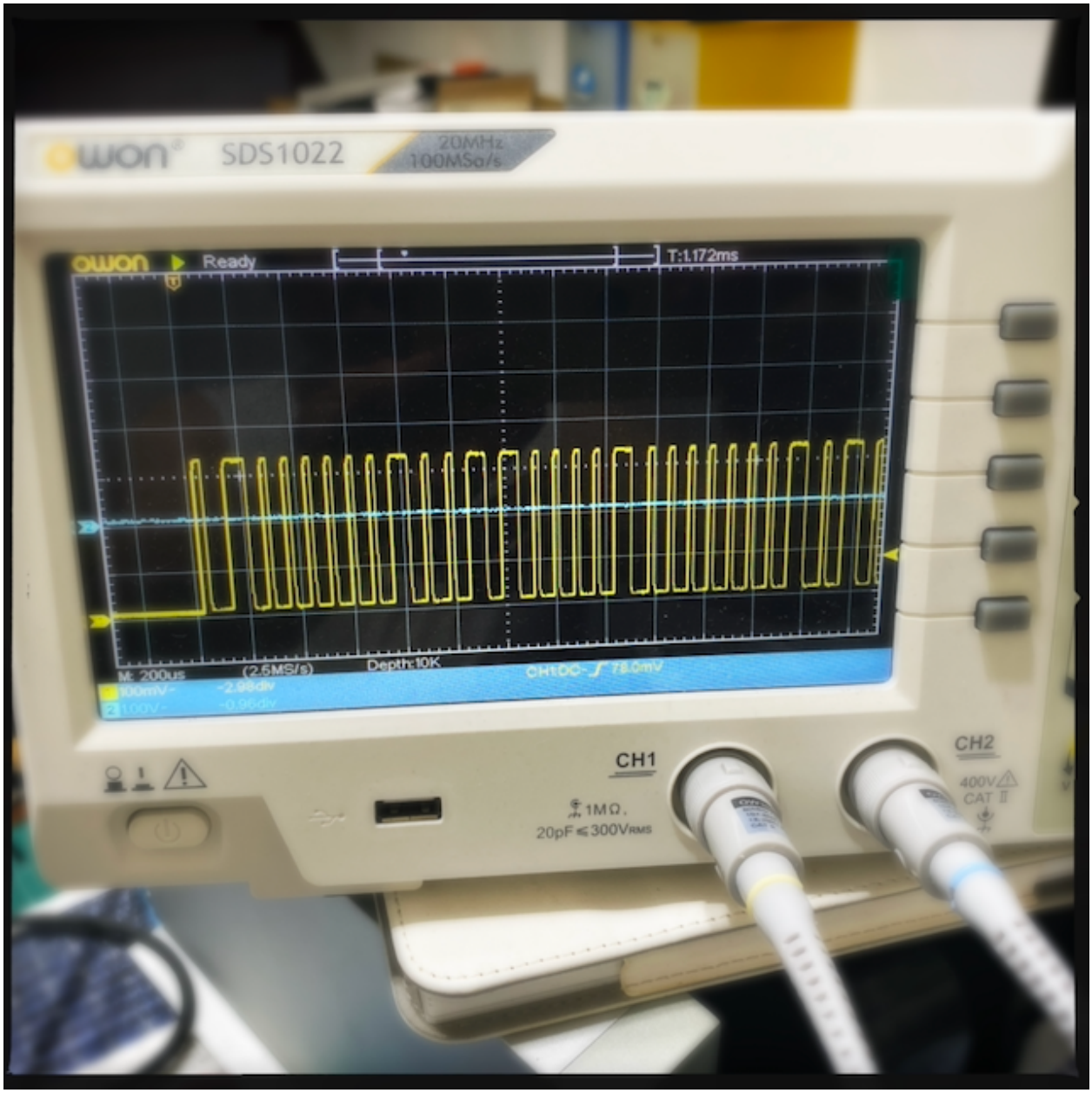
```
: 40bits@ ( -- n1 n2 n3 n4 n5)

  4 for 8bits@ next

;
```

Receive and decode 5 bytes of data in sequence and put them into the stack one by one, complete.

So executing the DHT@ command can request the DHT from 1-wire to transmit and receive the protocol well, 5 bytes data.

At this point, the hardware layer data communication and reading are completed. The rest is how to interpret the 5 bytes of data.

By the way, I send the photos measured by the oscilloscope. It can be clearly seen that the signal composed of a bunch of short and long pulse waves. Long pulse waves, representing 1; short pulse waves, representing 0.



Humidity, temperature data format for DHT11

The next step is to decode the 5 Bytes data. As mentioned earlier, the transmission protocol of the hardware layer of DHT11 and DHT22 is the same, and will eventually send 5 bytes out. But the "meaning" of these 5 bytes is different.

The format of DHT11 is: [humidity integer digits] [humidity decimal digits] [temperature integer digits] [temperature decimal digits] [CheckSum check code] a total of five digits

Wherein [CheckSum check code] = [humidity integer digits] + [humidity decimal digits] + [temperature integer digits] + [temperature decimal digits], it is used to check whether there are any transmission errors in the first four digits. If it is inconsistent, it means that the transmission is unstable, and some information is lost during the transmission process.

Therefore, when the five groups of numbers 67 0 31 0 98 are received, it represents

Humidity: 67.00 RH%

Temperature: 31.00 C

CheckSum = 67 + 0 + 31 + 0 = 98 is consistent with the fifth number, so the transmission is successful.

As mentioned earlier, the resolution of DHT11 is only 1% RH and 1C integer resolution, so the second decimal number representing humidity and the fourth decimal number representing temperature will always be zero.

So, the decoded code

```
: >DHT11 ( RHint RHdec Tint Tdec Checksum -- RH Temp)

   nip over - >r ( RHint RHdec Tint | R: checksum')

   nip over r> -

   abort" Error: CheckSum not match!!"

;
```

Tdec is the decimal place of the temperature, which is directly discarded by nip, leaving the temperature Tint in integer digits. Then CheckSum is subtracted from the integer bits of Tint.

RHdec is the decimal place of humidity, which is directly discarded by nip, leaving the humidity RHint of integer bits. CheckSum then continues to subtract the integer bits of RHint.

At this time, if the transmission is correct, CheckSum should become zero after two subtractions. If it is not zero, it means that there is an error in the transmission, an error message is sent and the Abort command is executed.

Humidity, temperature data format for DHT22

Next is the data format of DHT22, which is more complicated. So let's explain it directly with the example of the specification book.

DATA=16 bits RH data+16 bits Temperature data+8 bits check-sum

Example: MCU has received 40 bits data from AM2302 as

0000 0010 1000 1100 0000 0001 0101 1111 1110 1110

16 bits RH data 16 bits T data check sum

Here we convert 16 bits RH data from binary system to decimal system,

0000 0010 1000 1100 → 652

Binary system Decimal system

RH=652/10=65.2%RH

Here we convert 16 bits T data from binary system to decimal system,

0000 0001 0101 1111 → 351

Binary system Decimal system

T=351/10=35.1℃

When highest bit of temperature is 1, it means the temperature is below 0 degree Celsius.

Example: 1000 0000 0110 0101, T= minus 10.1℃

16 bits T data

Sum=0000 0010+1000 1100+0000 0001+0101 1111=1110 1110

Check-sum=the last 8 bits of Sum=1110 1110

The same is 5 Bytes: [RH-MSB] [RH-LSB] [T-MSB] [T-LSB] [CheckSum]

CheckSum is still [CheckSum] = [RH-MSB] + [RH-LSB] + [T-MSB] + [T-LSB], which is used to check whether there is any error in the transmission process.

But pay attention, CheckSum is only 1 byte, so it can only be up to 255, and the bit data exceeding 255 will be discarded.

Humidity = 1/10 * [RH-MSB]*256 + [RH-LSB] , that is, two bytes form a 16-bit single integer, and then an invisible decimal point.

So [0000 0010] [1000 1100] --> 652 means 65.2 % RH relative humidity

Temperature = 1/10 * [T-MSB]*256 + [T-LSB], that is, two bytes form a single integer of 16 bits, and then an invisible decimal point.

But, however, DHT22 can sense sub-zero temperatures between -40 - 0 yo. So the leftmost bit of the MSB represents the sign. Therefore, when it is a negative temperature, the leftmost bit should be removed first, so that the converted integer will be correct.

So [1000 0000] [0110 0101] = - [0000 0000] [0110 0101] = -101 --> -10.1 C temperature

So, **the decoded code**

```
: >DHT22 ( RH.H RH.L TH TL Checksum -- RH Temp)

   >r 2dup + >r

   swap 8 lshift or >r ( RH.H RH.LR: T CS1 CS)

   2dup + >r ( RH.H RH.LR: CS2 T CS1 CS)

   swap 8 lshift or ( RH R: CS2 T CS1 CS)

   r> r> swap ( RH T CS2 R: CS1 CS)

   r> + ( RH T CS3 R: CS)

   256 u/ mod drop

   r> <>

   abort" Error: CheckSum not match!!"


   $7fff over and swap ?negate

;
```

This code has more stacking operations and is less readable. At this time, a common technique used by FORTH programmers is to put a large number of annotations that change before and after the stack. After passing through these annotations, the readability is actually not that bad, and it is still very easy to maintain your code in the future.

Gradually add up the numbers of four bytes to check CheckSum. In the middle, it will be temporarily saved to the return stack through >r. When needed, move r> from the return stack back to the operation.

CheckSum is added in the final stage, the calculation is done, 256 u/mod drop to make it not exceed 256, so as to compare with the received CheckSum, if it is inconsistent, an error message will be sent and the Abort command will be executed.

8 lshift or is to combine [MSB] octets to the left followed by [LSB] and combine them into a single integer using an or operation.

The last important point is that since the temperature may be negative, the leftmost bit of the single integer will be 1 at this time. So $7fff over and removes this unsightly bit. When this bit is 1, the 2's complement system will consider it to be a negative number. So swap ?negate, when the number is negative, change the positive integer with the correct number to the "negative integer" of the correct 2's complement, and you are done.

So, the **complete DHT11 value code**

```
: DHT11@ ( -- RH T)

    DHT@ >DHT11

    10 * >r 10 * r>

;
```

DHT@ communicates with the DHT sensor through 1-wire to request data, and then converts the data into the correct humidity and temperature to check CheckSum for errors. Then, in order to be consistent with the subsequent DHT22 format, the temperature and humidity are multiplied by 10 times. (meaning one decimal place)

Complete **DHT22 value code**

```
: DHT22@ ( -- RH T)

    DHT@ >DHT22

;
```

DHT@ communicates with the DHT sensor through 1-wire to request data, and then converts the data into the correct humidity and temperature to check CheckSum for errors.

Integer with sign and one decimal place is printed

```
: .[xx.x] dup <# # [char] . hold #s swap sign #> type ;
```

Note that the ESP32FORTH is a 32-bit system. 1 cell integer is 32 bit, so ESP32FORTH here <# is a bit substandard, not double integer with 2 cells. So there is no need to convert single integer to double integer s>d

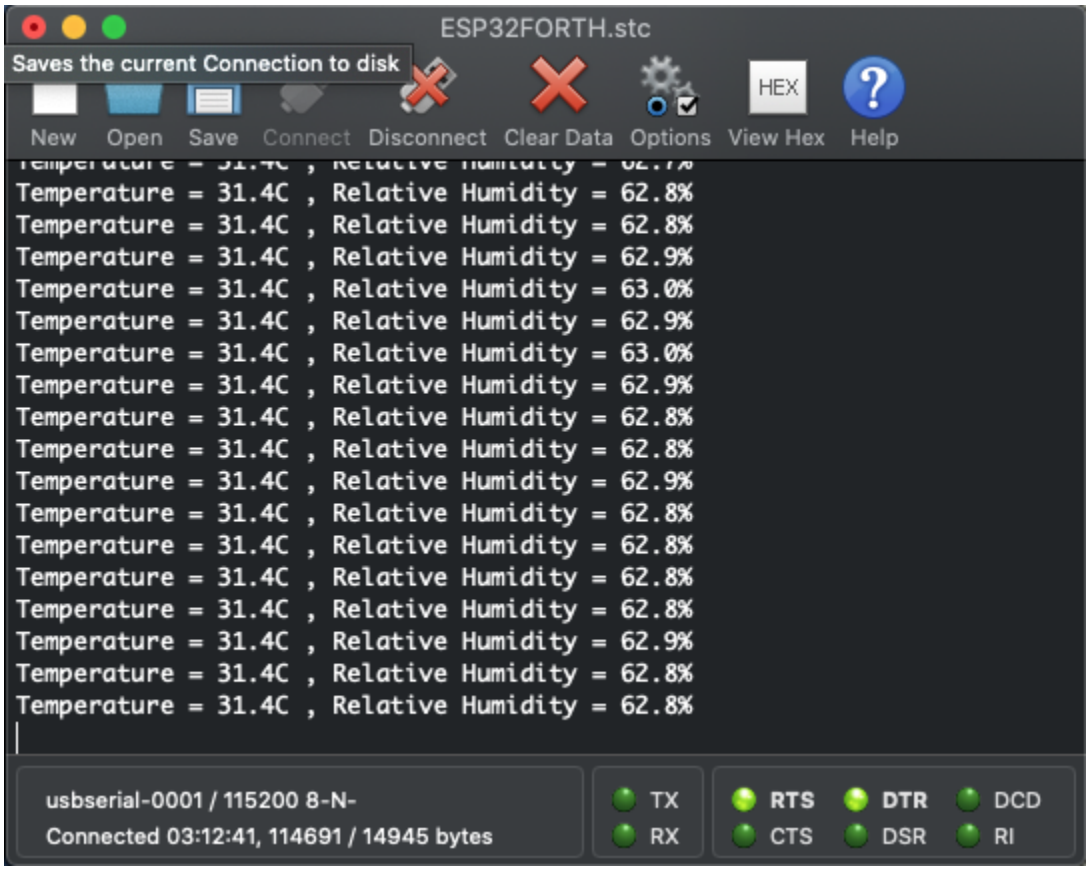
Finally, keep printing the value.

```
: DHT11
   cr
   begin
     DHT11@
     ." Temperature = " .[xx.x] ." C , "
     ." Relative Humidity = " .[xx.x] ." %" cr
     2000 ms
   again
;

: DHT22
   cr
   begin
     DHT22@
     ." Temperature = " .[xx.x] ." C , "
     ." Relative Humidity = " .[xx.x] ." %" cr
     2000 ms
   again
;
```

Results of the

The result of the author's three DHT11s is that the temperature is 15 degrees and the humidity is 18%, which are very outrageous values. They should all be broken.

But DHT22 is great, very stable, 31.6C and relative humidity around 62% on a hot summer night.



Then as long as the sensor is deliberately unplugged, it will be Abort immediately because the pulse is wrong. It really works perfectly! 😭

Finally, the full source code listing

```
\ DHT11, DHT22 1-wire Control Code - ESP32FORTH
\ Frank Lin 2022.7.20
\
\ Digital I/O Access Codes
\

:>OUTPUT ( pin --) \ set the direction of digital I/O to output
    output pinMode
;
: <INPUT ( pin --) \ set the direction of digital I/O to input
    input pinMode
;
: PULLUP ; immediate \ dummy for syntax sweeter
: ->High ( pin --) \ put digital I/O to High
   high high digitalWrite
;
: ->Low ( pin --) \ put digital I/O to Low
   low digitalWrite
;
: Pin@ ( pin -- status) \ read the state of digital I/O, 0=low, 1=high
   digitalRead
;
: ticks ( -- ticks )
   ms-ticks
;
```

```
\
\ extension for ESP32FORTH
\ ESP32FORTH only has catch/throw, no standard ABORT, ABORT"
\

: abort ( --) -1 throw ;
: abort" ( flag "text" --)
   state @
   if
      postpone if postpone s" postpone type postpone cr
                     postpone abort
      postpone then
   else [char] " parse type cr abort then
; immediate



\
\ DHT Sensor, 1-wire data Pin
\

14 constant DHTPin \ Pin14 as DHT data Pin

: delay ( n--) for next ; \ used as the delay timer

\
```

```
\ DIO and delay speed test
\
: t1 ticks DHTPin 1000000 for DHTPin Pin@ 0= if then next drop ticks swap - .
;
\
\ result: 642 ms / 1000000 = 0.642 uS per loop
\


: t2 ticks 1000000 delay ticks swap - . ;
\
\ result: 96.5 ms / 1000000 = 0.0965 uS for 1 delay
\


: wait ( --) \ wait until pulse-high
   begin DHTPin Pin@ until
;


\
\ DHT 1-wire signal:
\ start: 50uS Low
\ signal 1: 70 uS Pulse High
\ signal 0: 26 - 28 uS Pulse High


\
\ 112uS / 0.642uS = 174
\ 67.175uS / 0.642uS = 104
```

```
\


: signal@ ( -- true=1/false=0)
   174 ( ~112uS)
   for
       DHTPin Pin@ 0= ( pulse low?)
       if r> 104 ( ~ 67.175uS) < exit ( length > 70 = 44.825 uS)
       then
   next
   ." Error! Signal not match with expectation!" cr
   abort
;


: 8bits@ ( -- Data)
   0 ( data)
   7 for
        wait
        signal@ if 1 r@ lshift or then
     next
;
: 40bits@ ( -- n1 n2 n3 n4 n5)
  4 for 8bits@ next
;
\
\ Start Signal
\ 18mS Low, to active communication
```

```
\ then 20 - 40uS High
\ then wait DHT sends 80uS Low, 80uS High
\ then receive 40bits data transmition from DHT
\
\
\ 20uS = 20/0.0965 ~ 207 delay
\ 82uS = 82/0.0965 ~ 850 delay
\ 14uS = 14/0.0965 ~ 145 delay
\
: start! ( --)
   DHTPin >OUTPUT
   DHTPin ->Low
     20 ms
   DHTPin ->High
     145 delay ( ~ 14uS)
   DHTPin <INPUT

;
: DHT@ ( -- n1 n2 n3 n4 CheckSum)
   start!
   207 delay ( ~ 20uS)
   wait
   850 delay ( ~ 82uS) 40bits
   @
;
```

```
: >DHT11 ( RHint RHdec Tint Tdec Checksum -- RH Temp)
   nip over - >r ( RHint RHdec Tint | R: checksum')
   nip over r> -
   abort" Error: CheckSum not match!!"
;

: ?negate ( n1 n2 -- n3)
    $80 and if negate then
;

: >DHT22 ( RH.H RH.L TH TL Checksum -- RH Temp)
   >r 2dup + >r
   swap 8 lshift or >r ( RH.H RH.LR: T CS1 CS)
   2dup + >r ( RH.H RH.LR: CS2 T CS1 CS)
   swap 8 lshift or ( RH R: CS2 T CS1 CS)
   r> r> swap ( RH T CS2 R: CS1 CS)
   r> + ( RH T CS3 R: CS)
   256 u/ mod drop
   r> <>
   abort" Error: CheckSum not match!!"

   $7fff over and swap ?negate
;
: DHT11@ ( -- RH T)
    DHT@ >DHT11
    10 * >r 10 * r>
```

```
;

: DHT22@ ( -- RH T)

    DHT@ >DHT22

;


: .[xx.x] dup <# # [char] . hold #s swap sign #> type ;


: DHT11

   cr

   begin

     DHT11@

     ." Temperature = " .[xx.x] ." C , "

     ." Relative Humidity = " .[xx.x] ." %" cr

     2000 ms

   again

;


: DHT22

   cr

   begin

     DHT22@

     ." Temperature = " .[xx.x] ." C , "

     ." Relative Humidity = " .[xx.x] ." %" cr

     2000 ms

   again

;
```