Specyfikacja projektu

Płytka: Nucleo-F401RE ARM Cortex M4

1. Opracować komunikację mikroprocesora z PC poprzez interfejs asynchroniczny z wykorzystaniem przerwań i buforów kołowych.

2. Zaprojektować i zaimplementować protokół komunikacji pozwalający na adresowanie danych z uwzględnieniem ich kolejności.

3. Obsługa czujnika AHT15 poprzez interfejs I2C pracujący z przerwaniami
4. Dane pobierane co zadany interwał czasowy zdefiniowany w milisekundach i zapisywany do bufora kołowego o pojemności 1000 wpisów
5. Możliwość podglądu wartości bieżących i archiwalnych.

Projekt protokołu komunikacyjnego

Ramka:

Początek ramki	**Nadawca**	**Odbiorca**	**Komenda**	**Długość danych**	**Dane**	**CRC16**	**Koniec ramki**
0x3A **“ : ”**	Dowolne dane	Dowolne dane	Dowolne dane	Liczba z przedziału od 0 do 99.	Dane zapisane na podstawie bajtów.	Wszystkie cyfry zapisane heksadecymalnie	**0x3B** “ ; ”
1 bajt	1 bajt	1 bajt	1 bajt	1 bajt	0 - 99 bajtów	4 bajty	1 bajt

Opis danych w ramce:

Początek i koniec ramki:	Początkiem ramki jest znak „:”, a znakiem końca jest “ ; ”. Określa od jakiego miejsca ramka powinna być odczytywana, oraz gdzie powinien kończyć się odczyt ramki.
Nadawca i odbiorca:	Identyfikator nadawcy i odbiorcy o długości 1 bajtu, zawierającego informację z jakiego urządzenia przekazywana jest ramka i jakie urządzenie ma ją odebrać. Każdy bajt zapisany w.
Komenda:	Identyfikator komendy o długości 1 bajtu, zawierający informację jaka komenda ma zostać wykonana.
Długość danych	Długość danych jest zapisana na w liczbie o długości 1 bajtu. Ale nie może przekraczać zawartości od 0 do 99. Czyli musi się zgadzać z długością pola dane. Bajt zapisany za pomocą.
Dane:	Ciąg bajtów zakodowanych które są zbierane w całość bajt po bajcie. Czyli jak wyślemy liczbę 16 bitową to zostanie złożona z dwóch bajtów. A jak będziemy chcieli przesłać wartość tekstową w kodowaniu to będzie odczytywane bajt po bajcie. Konwencja sprawdzania bajtów jest to.
CRC16:	Pole CRC16 zawiera sumę kontrolną o długości 4 bajtów, wartość bajtu jest reprezentowana przez odpowiadający jej znak latian-1 reprezentujących wartość 16-bitową w systemie szesnastkowym. Przykładowo, dla `0xA3F2` zapis w ramce to `A3F2`. **Dodatkowo to pole nie jest kodowane w base64.**

Jak wygląda przykładowe wysyłanie ramki:

Na koniec wszystkie pola **poza sumom kontrolną** są kodowane do Base64 z opcją paddingowania znaków do 4 bajtów. I ostateczna wysyłana ramka wygląda następująco : Nagłówek_Base64 CRC16-modbus ;

Przykładowy wysłana ramka:

Chcemy wysłać podaną ramkę -> **:Pj8mBAAAB9A=F8FA;**
Ta ramka dzieli się na -> znak początku -> **:**
Zakodowany w base64 znak ->

1. Nadawcy
2. Odbiorcy
3. Komendy
4. długości danych
5. pola dane -> **Pj8mBAAAB9A=**

Znaki heksa decymalne reprezentujące sumę kontrolną, te znaki nie odpowiadają reprezentacji bitowej np. znaku F czyli 1111, tylko to są znaki zapisane w dodowaniu ASCII reprezentujące w postaci hexadecymalnej liczbę sumy kontrolnej -> **F8FA**
Znak końca ramki -> **;**

Kodowanie za pomocą Base64:

Sposób kodowania:

Text content	M								a								N							
ASCII	77								97								110							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19						22						5				46							
Base64-encoded	T						W						F				U							

Tabela do kodowania czyli zbiór bitów do których jest przypisany dana liczba:

Uwaga!!!
Padding którego używam do wypełnienia kodowania by składał się ze znaków o ilości wielokrotności 4. Ponieważ moje kodowanie przewiduje że możliwe jest wysłanie 1, 2 lub 3 bajtów ale wtedy trzeba dodać na końcu znak „=”. Jest to znak który jest inaczej odbierany niż inne znaki z kodowanie Base64. Ten znak odpowiada znakowi kodowania z tablicy latian-1 czyli jest to wartość 75. To ma umożliwić mi określenie ile bajtów znajduje się w danej wielokrotności 4. Dzięki temu wiem że jak zostanie wysłana wartość:

1. /w== -> to wiem że został wysłany 1 bajt, który składa się z samych jedynek.
2. /wA= -> to wiem że zostały wysłane 2 bajty, które składają się z jednego bajtu złożonego z samych jedynek i drugiego bajtu składającego się z samych zer.
3. /wD/ -> to wiem że zostały wysłane 3 bajty, które składają się z jednego bajtu złożonego z samych jedynek i drugiego bajtu składającego się z samych zer i trzeciego bajtu złożonego znowu z samych jedyne.

Wartość padding **nie odgrywa żadnej roli podczas składania bajtów**. Jest ona stałą wartością którą nie wykorzystujemy do innych celów jak tylko do tego by nasz ciąg bajtów zakodowanych w base64 był wielokrotnością 4.

Zapis decymalny	**Zapis binarny**	**Znak kodujący**
0	000000	A
1	000001	B
2	000010	C
3	000011	D
4	000100	E
5	000101	F
6	000110	G
7	000111	H
8	001000	I
9	001001	J
10	001010	K
11	001011	L

12	001100	M
13	001101	N
14	001110	O
15	001111	P
16	10000	Q
17	10001	R
18	10010	S
19	10011	T
20	10100	U
21	10101	V
22	10110	W
23	10111	X
24	11000	Y
25	11001	Z
26	11000	a
27	11001	b
28	11010	c
29	11011	d
30	11100	e
31	11101	f
32	11110	g
33	11111	h
34	100000	i
35	100001	j

36	100010	k
37	100011	l
38	100100	m
39	100101	n
40	100110	o
41	100111	p
42	101000	q
43	101001	r
44	101010	s
45	101011	t
46	101100	u
47	101101	v
48	101110	w
49	101111	x
50	110000	y

51	110001	z
52	110100	0
53	110101	1
54	110110	2
55	110111	3
56	111000	4
57	111001	5
58	111010	6
59	111011	7
60	111100	8
61	111101	9
62	111110	+
63	111111	/

Kodowanie znaków użyte w moim projekcie:

Użyłem kodowaniania **latian-1** ponieważ on zapisuje dane w postaci **jednego bajtu** a **nie na 4 bajtach** jak to ma miejsce w **utf-8.** A muszę wprowadzić rozróżnienie między tymi dwoma sposobami kodowania ponieważ ja przesyłam wszystkie znaki ale w **tylko w postaci jednego bajtu** gdybyby znak był zapisany do większej ilości niż 1 bajt to stanowił by to problem dla mojego protokołu. A mimo że utf-8 wspiera kodowanie ascii to jednakże tylko do pewnego stopnia.

Cytaty z Wikipedii:
Przy pomocy ISO 8859-1 można zakodować to, co zwane jest alfabetem łacińskim numer 1, który składa się ze 191 znaków pisma łacińskiego. Każdy znak jest kodowany jako pojedyncza **8-bitowa wartość**.

UTF-8 (ang. *8-bit Unicode Transformation Format*) – system kodowania Unicode, wykorzystujący **od 1 do 4 bajtów do zakodowania pojedynczego znaku**, w pełni kompatybilny z ASCII. Jest najczęściej wykorzystywany do przechowywania napisów w plikach i komunikacji sieciowej.

Sposób kodowania
Mapowanie znaków Unicode na ciągi bajtów możliwe jest na jeden z czterech sposobów:

- U+00 do U+7F – **0**xxxxxxx, gdzie kolejne „x” to bity od najstarszego,
- U+80 do U+7FF – **110**xxxxx **10**xxxxxx,
- U+800 do U+FFFF – **1110**xxxx **10**xxxxxx **10**xxxxxx,
- U+10000 do U+10FFFF – **11110**xxx **10**xxxxxx **10**xxxxxx **10**xxxxxx.

Znaki z przedziału ASCII (0 do 127) kodowane są jako jeden bajt, czyli m.in. litery alfabetu łacińskiego. Polskie litery diakrytyzowane kodowane już są jako dwa bajty.

Tabela znaków przypisanych do komend:

Opis wykonywanie komendy	Znak przedstawiający podaną komendę	Parametry komendy	Typ
Inicjalizacja czujnika	!	bajt z samymi bitami o wartości 0.	Uint8_t
Odpowiedź z podanej komendy ->	!	Przesłana odpowiedź gdy czujnik został poprawnie zainicjalizowany: Odpowiedz: Czujnik AHT15 poprawnie zainicjalizowany.	
Włączanie pomiaru danych w określonym czasie.	$	bajt z samymi bitami o wartości 0.	Uint8_t
Odpowiedź z podanej komendy ->	$	Przesłana odpowiedź: Odpowiedz: Włączenie pomiaru cyklicznego.	

Komenda kończąca wykonywanie interwału pobierania danych z czujnika.	%	bajt z samymi bitami o wartości 0.	Uint8_t
Odpowiedź z podanej komendy ->	%	Przesyłana odpowiedź: Odpowiedz: Zakończenie pobierania co interwał	
Ustawianie interwału	&	Wpisujemy tu liczbę 32 bitową reprezentującą interwał pobierania danych z czujnika. Czyli jak będziemy chcieli wysłać liczbę 2000 to wyślemy 4 bajty w postaci 32 bitowej czyli 0000 0000 0000 0111 1101 0000 jest to uzupełniane zerami po to by zachować strukturę przewidzianą właśnie dla podanej komendy. Podaną wartość zapisujemy w little endian.	Uint_32
Odpowiedź z podanej komendy ->	&	Przesłana odpowiedź: Odpowiedz: Ustawianie interwału.	
Komenda pobierająca określoną ilość danych archiwalnych z czujnika.	^	Podana komenda zawiera podane parametry indeks od którego chcemy pobrać liczbę i ilość pobieranych danych. Dane żeby były pobrane to wysyłamy w podanej formie **indexilość.** Tu możemy na przykład wysłać dwie liczby 16 bitowe zaraz występujące po sobie. Przykład: chcemy pobrać 10 wartości z bufora od indexu 5. Więc wysyłamy: 0000 0000 0000 0101 0000 0000 0000 1010 W przypadku liczb mniejszych niż 16 bitów to muszą być wypełnione zerami. Jest to konieczne by zachować odpowiednią strukturę pobierania liczb. Dodatkowo liczba indexu będzie liczona od 0 do 999 a liczba pobieranych wartości będzie z zakresu od 1 do 1000. Podane wartości zapisujemy w little endian.	Uint16_tUint16_t
Odpowiedź z podanej komendy ->	^	Przesłana odpowiedź: Przesyłam pomiar temperatury i wilgorności i inedx w podanej kolejności: Uint_32 Uint_32Uint_16 Wartości temperatury będzie wysyłana binarnie w formacie **IEEE 754.** Uwaga!! Podany format odpowiedzi jest przewidziany dla jednego pomiaru ale jak w przypadku komendy ^ może zostać pobranych wiele pomiarów to żeby nie wysyłać każdego pomiaru w osobnej ramce to wysyłam tyle pomiarów ile w danym momęcie zmieści się do mojej ramki czyli 99 bajtów. Więc w jednej ramkce w polu dane może się zawierać więcej niż jeden pomiar. Przez to że konieczne będzie wysyłanie wielu ramek to też na samym końcu mojej komendy jest napisany index który to jest odczytany pomiar tak aby jakby jakaś ramka się zgubła to można by było określić jaka ramka się zgubiła.	
Komenda pobierająca aktualną wartość danych z czujnika.	*	bajt z samymi bitami o wartości 0.	Uint8_t
Odpowiedź z podanej komendy ->	*	Przesłana odpowiedź : Przesyłam pomiar temperatury i wilgorności i inedx w podanej kolejności, bity są ułożone w formacie little endian: Uint_32 Uint_32Uint_16 Wartości temperatury będzie wysyłana binarnie w formacie **IEEE 754.** Wartość indexu będzie równa w tym przypadku 5000 by oznaczało to że wartość została pobrana manualnie.	
Komenda pobierająca jaki czas został ustawiony jako interwał.	@	bajt z samymi bitami o wartości 0.	Uint8_t
Odpowiedź z podanej komendy ->	@	Przesłana odpowiedź: Przesyłam ustwiony interwał w podanym formacie little endian: Uint_32	
Komenda mająca za zadanie wskazanie że została wysłana ogólną odpowiedź.	G	Dane z odpowiedzi z mikrokontrolera. Ta komenda wskazuje czy że ma tylko za cel informacyjny bez wywoływania żadnego efektu, po stronie PC.	Uint8_t

Wysyłana odpowiedź:

Wysyłamy odpowiedź w formie naszego protokołu komunikacyjnego ale gdy wysyłana odpowiedź przekracza 99 bajtów to wtedy jest dzielona na pozostałe ramki.

Przykład odebranej odpowiedzi:

:Pz5HIE9kcG93aWVkejogVXN0YXdpYW5pZSBpbnRlcndhbHUuCBF7;

- Składa się ze znaku początku: **:**
- Zakodowanego w base64:
  - Nadawca: ?
  - Odbiorca: >
  - Komenda: G
  - Długość danych:

32	040	20	00100000	SP		Space

-> czyli 32 w zapisie dziesiętnym.

- Pole dane: Odpowiedz: Ustawianie interwalu.

- Suma kontrolna crc16 – modbus: **CBF7**
- Znak końca: **;**

❖ **Wysłanie komendy rozpoczynającej pomiar:**
- ➢ Wysyłam komendę wywołującą rozpoczęcie pomiaru.
- ➢ Czujnik rozpoczyna proces pomiaru.

❖ **Odczyt danych pomiarowych:**
- ➢ Gdy pomiar jest gotowy (status bit [Bit7] = 0), mikrokontroler ponownie wysyła adres czujnika, ale tym razem z bitem odczytu (R: 1).
- ➢ Płytka wysyła odebrane dane które zostały najpierw obliczone za pomocą wzoru:
  - ■ **Wilgotność**:
    - Dane wilgotności zajmują **pierwsze 20 bitów**.
    - Konwersja wyniku:
      - ◆ Łącze ze sobą pierwsze 20 bitów z przesłanego odbioru danych.
      - ◆ Obliczam dane za pomocą wzoru:

$$\mathrm{RH}[\%] = \left(\frac{\mathrm{S}_{RH}}{2^{20}}\right) * 100\%$$

      - ◆ gdzie Srh to połączone dane wilgotności zapisane w formie liczby dziesiętnej.
      - ◆ Dane są podawane w (Dopuszczalnej względnej wilgotność otoczenia (RH – Relative Humidity) to wartość wyrażona w procentach (0–100%))

  - ■ **Dane temperatury** które zajmują kolejne **20 bitów**.
    - Konwersja wyniku:
      - Łączenie ze sobą bitów.
      - ◆ Obliczam dane za pomocą wzoru:
      - ◆

$$\mathrm{T(C)} = \left(\frac{\mathrm{S}_{T}}{2^{20}}\right) * 200 - 50$$

      - ◆ gdzie St to połączone dane temperatury zapisane w formie liczby dziesiętnej.
      - ◆ Temperatura będzie wyrażona w Celsjuszach.

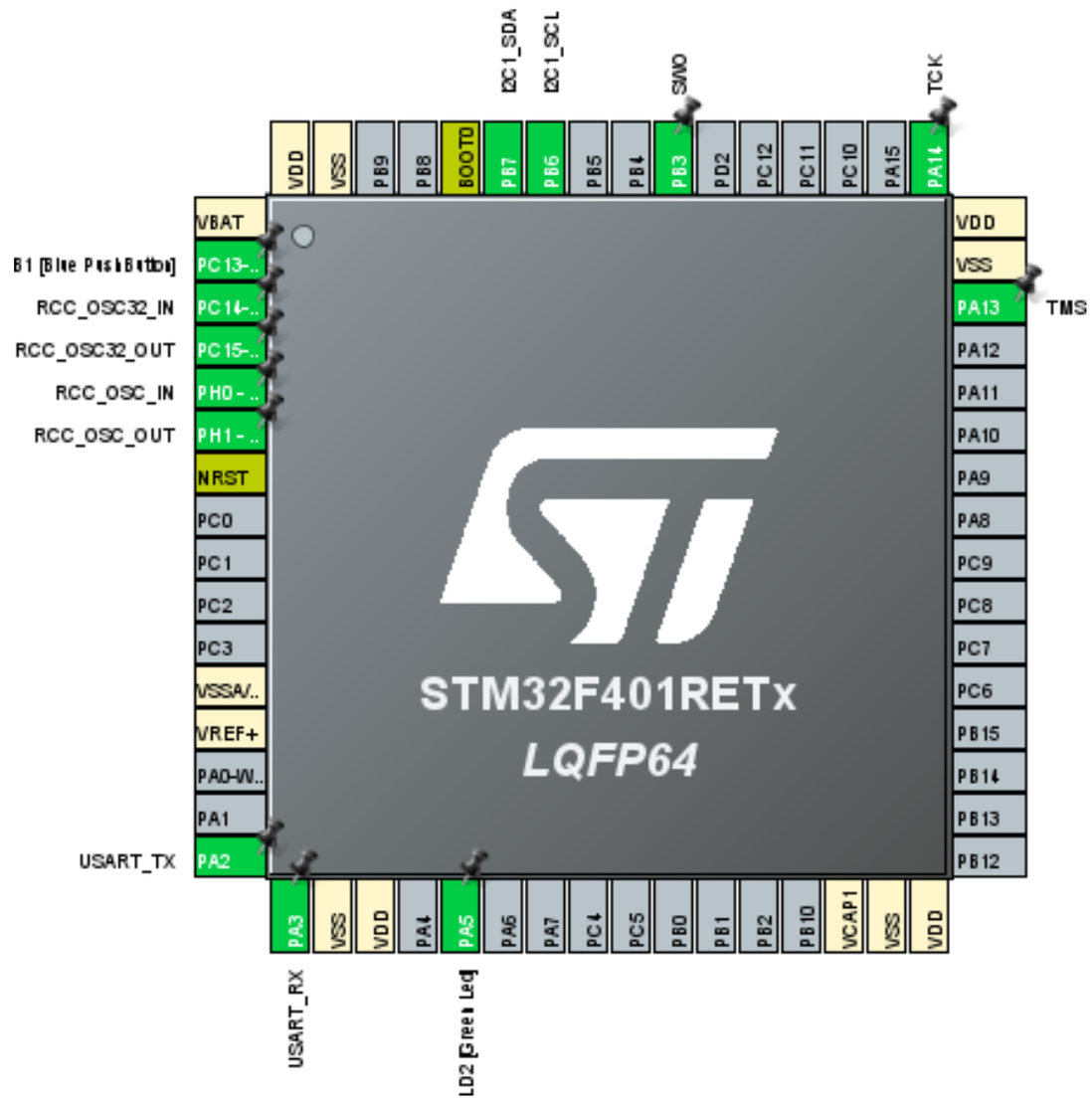
- Zapisuje pobrane dane i wyliczone za pomocy podanych wzorów do bufora kołowego w kodowaniu ASCII w formacie liczby dziesiętne j.

Obsługa błędów:

Opis	Akcja	Informacje zwrotne
Brak znaku końca ramki	Zignorowanie ramki	Brak
Brak znaku początku ramki	Zignorowanie ramki	Brak
Nieprawidłowy odbiorca	Zignorowanie ramki	Brak
Nieprawidłowy nadawca	Zignorowanie ramki	brak
Znaki przed znakiem początku lub po znaku końca ramki	Zignorowanie ramki	Brak
Wiele znaków początki lub/i końca ramki	Rozpatrywanie ramki od ostatniego znaku początku ramki	Brak
Niepowodzenie podczas sprawdzania crc16	Zwrócenie błędu i Zignorowanie ramki	Zwraca ramkę o komendzie ANSWER i danych „Niepowodzenie podczas

Konfiguracja mikrokontrolera:

Ustawienie pinów w mikrokontrolerze:



Konfiguracja USART:

Parametry:

Bund Rate	115200 Bits/s

Word Length	8 Bits
Partity	None
Stop Bits	1

Specyfikacja czujnika:

Parametr	Wartość
Napięcie zasilania	1.8–3.6 V (3.3 V zalecane)
Pobór prądu	23 μA (pomiar), 0.25 μA (tryb uśpienia)
Moc	0.07 mW (pomiar), 0.9 μW (uśpienie)

Parametr	Temperatura	Wilgotność	Jednostka
Rozdzielczość	0.01	0.024	°C / %RH
Dokładność	±0.3	±2	°C / %RH
Powtarzalność	±0.1	±0.1	°C / %RH
Histereza	±0.1	±1	°C / %RH
Czas odpowiedzi	5–30	8	s
Zakres pomiarowy	-40–85	0–100 (Normalny zakres roboczy wynosi: 0-80% RH	°C / %RH
Dryft	<0.04/rok	<0.5/rok	°C/yr

Zalecenia dotyczące spawania:

Zabrania się stosowania lutowania rozpływowego lub falowego. Lutowanie ręczne dynamiczne musi być wykonywane w temperaturach do 350°C przez mniej niż 5 sekund.

Uwaga: Po spawaniu czujnik musi znajdować się w otoczeniu o wilgotności względnej >75% przez co najmniej 12 godzin, aby zapewnić ponowną polimeryzację/hydratację. Niezastosowanie się do tego zalecenia spowoduje dryft odczytu czujnika. Należy również umieścić czujnik w naturalnym środowisku (o wilgotności względnej >40%) na 5 dni lub dłużej, aby umożliwić jego ponowne nawodnienie. Użycie lutu o niskiej temperaturze (np. 180°C) może skrócić czas hydr atacji.

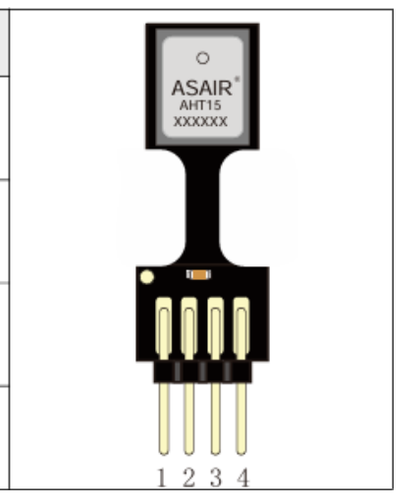
Jeśli czujnik jest stosowany w obecności gazów korozyjnych lub występuje kondensacja wody (np. w środowisku o wysokiej wilgotności), pady ołowiane i PCB należy uszczelnić (np. za pomocą powłoki konformalnej), aby uniknąć słabego kontaktu lub zwarcia.

Zasady okablowania i integralność sygnału:

Jeśli linie sygnałowe SCL i SDA są równoległe do siebie i są ze sobą połączone bardzo blisko, może to powodować przesłuch sygnału i awarię komunikacji. Rozwiązaniem jest umieszczenie VDD i/lub GND między tymi dwoma liniami sygnałowymi, oddzielenie linii sygnałowych i użycie kabli ekranowanych. Dodatkowo, obniżenie częstotliwości SCL może również poprawić integralność transmisji sygnału. Należy dodać kondensator tantalowy 100nF między pinami zasilania (VDD, GND) w celu filtrowania.
Kondensator ten powinien znajdować się jak najbliżej czujnika.

Definicja interfejsu:

Pin	name	Interpretation
1	VDD	Supply（1.8-3.6V）
2	SDA	Data ，2-way
3	GND	Ground
4	SCL	Clock ，2-way



Pin zasilania (VDD, GND)

Zakres napięcia zasilania AHT15 wynosi 1,8-3,6 V, zalecane napięcie to 3,3 V. Między zasilaniem (VDD) a masą (GND) należy podłączyć kondensator odsprzęgający 100nF, umieszczając go jak najbliżej czujnika.

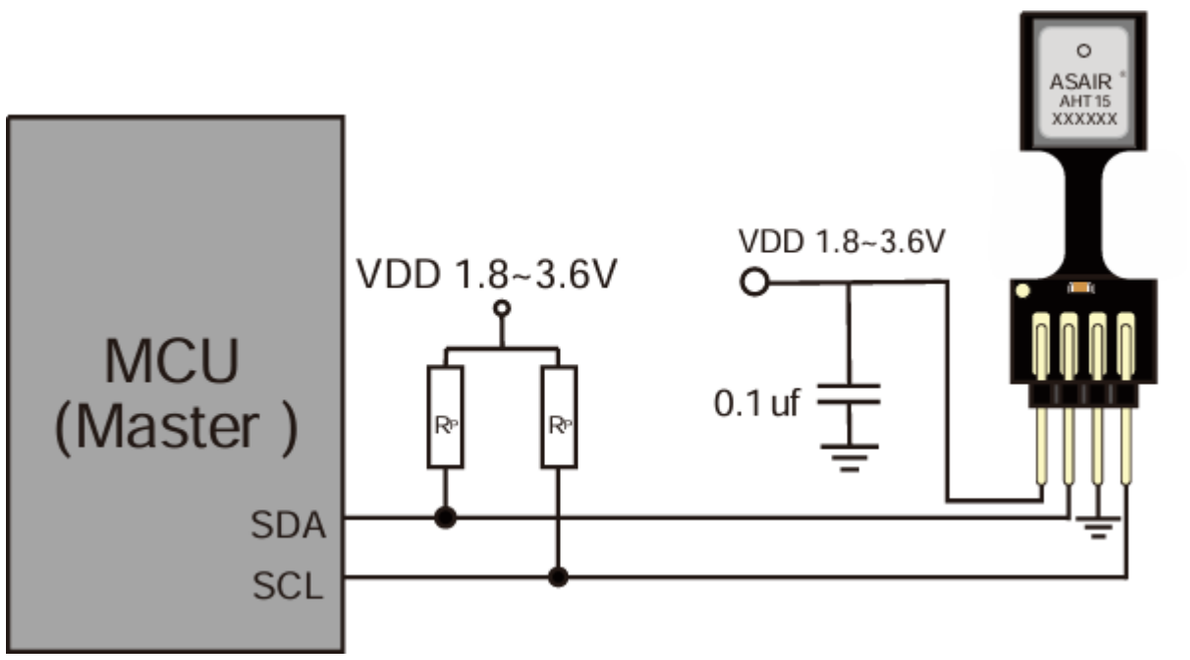
Uwaga:

1. Napięcie zasilania MCU hosta musi być zgodne z napięciem czujnika podczas użytkowania produktu.
2. Aby uniknąć kolizji sygnałów, mikroprocesor (MCU) może sterować SDA i SCL tylko w stanie niskim. Potrzebny jest zewnętrzny rezystor podciągający AHT15, który podciąga sygnał do poziomu wysokiego. Rezystory podciągające mogą być zazwyczaj zawarte w obwodzie I/O mikroprocesora.

Szeregowe dane SDA

SDA jest ważny na rosnącym zboczu zegara szeregowego (SCL) podczas wysyłania poleceń do czujnika, a gdy SCL jest w stanie wysokim, SDA musi pozostać stabilne. Po opadającym zboczu SCL wartość SDA może zostać zmieniona. Aby zapewnić pełną komunikację, efektywny czas SDA przypada przed rosnącym zboczem SCL i powinien być przedłużony do TSU i THO po opadającym zboczu. Podczas odczytywania danych z czujnika, SDA jest aktywne (TV) po przejściu SCL w stan niski i pozostaje w tym stanie do następnego opadającego zbocza SCL.

Przykładowe podłączenie czujnika:



Komunikacja z czujnikiem

Uruchamianie czujnika

Napięcie zasilania VDD (z zakresu od 1,8 V do 3,6 V). Po włączeniu zasilania czujnik potrzebuje do 20 milisekund (w tym czasie SCL jest w stanie wysokim), aby osiągnąć stan spoczynku, czyli stan gotowości do odbierania poleceń wysyłanych przez hosta (MCU).

Wysyłanie poleceń

Po zainicjowaniu transmisji, kolejne dwa bajty I2C obejmujące pierwszy bajt z 7-bitowym adresem urządzenia I2C 0x38 i bit kierunku SDA (odczyt R: '1', zapis W: '0'). Po ósmym opadającym zboczu zegara SCL, pin SDA jest ściągany do stanu niskiego (bit ACK), wskazując, że odbiór danych przez czujnik jest normalny.

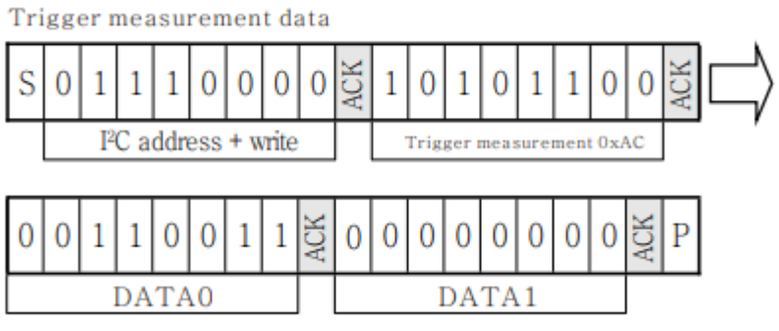
Lista komend i adresu urządzenia:

Command	code
Initialization	0xE1
Measurment trigger	0xAC
Soft reset	0xBA
I2C device addres	0x38

Opis bajtu stanu, ten bajt stanu zawiera precyzyjne informacje o aktualnym stanie czujnika.

Bit	Significante	description
Bit[7]	Busy (Busy indication) Sprawdza czy czujnik nie jest zajęty.	1 – Device is busy 0 – idle, sleep state
Bit[6:5]	Working mode (Mode Status)	00 NOR mode 01 CYC mode 1x CMD mode
Bit[4]	Reserved	Reserved
Bit[3]	Calibration enable (CAL Enable) Ten bajt odpowiada za to czy czujnik jest aktualnie skalibrowany do wykonywania pomiarów.	1 - calibrated 0 - not calibrated
Bit[2 : 0]	Reserved	Reserved

Wysyłanie komendy będzie się odbywało na czterech bajtach tak jak jest pokazane na obrazku z dokumentacji czujnika:

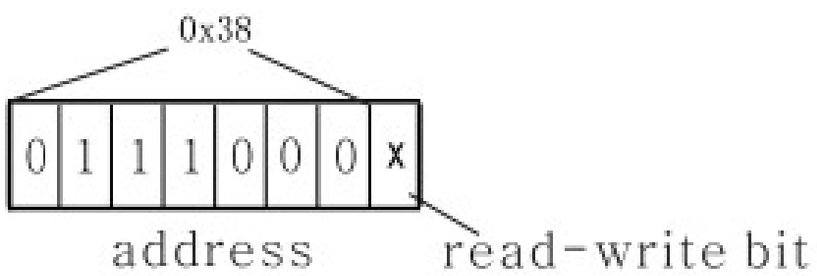


Ten obrazek pokazuje przykładowe wysłanie komendy odpowiedzialnej za uruchomienie wykonania pomiaru czujnika. Jak na tym obrazku można zobaczyć to pierwszy bajt odpowiada za przekazanie adresu czujnika który na końcu ma ostatni bit informujący czy jest to zapis czy odczyt. Potem wysyłamy kolejne bajty danych.

Uwaga!!! Ostatni bit w adresie czujnika trzeba odpowiednio dopasować w zależności od wysyłanej komendy wysyłanej komendy.

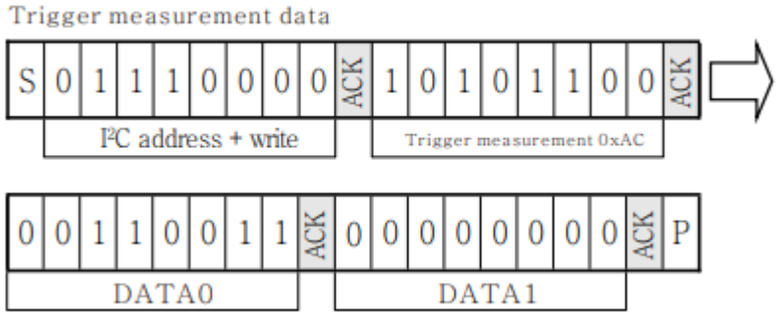
Żeby zainicjalizować czujnik to trzeba mu wysłać podane bajty:

1. Bajt z adresem czujnika i z ostatnim bajtem zapisu lub odczytu, w takim ułożeniu bajtów jakim jakie jest pokazane na podanym obrazku:

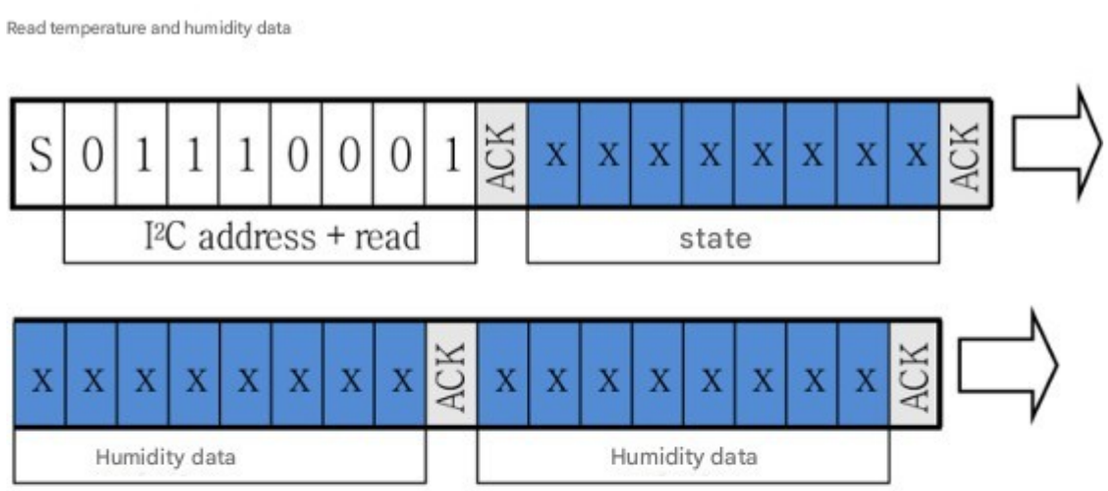


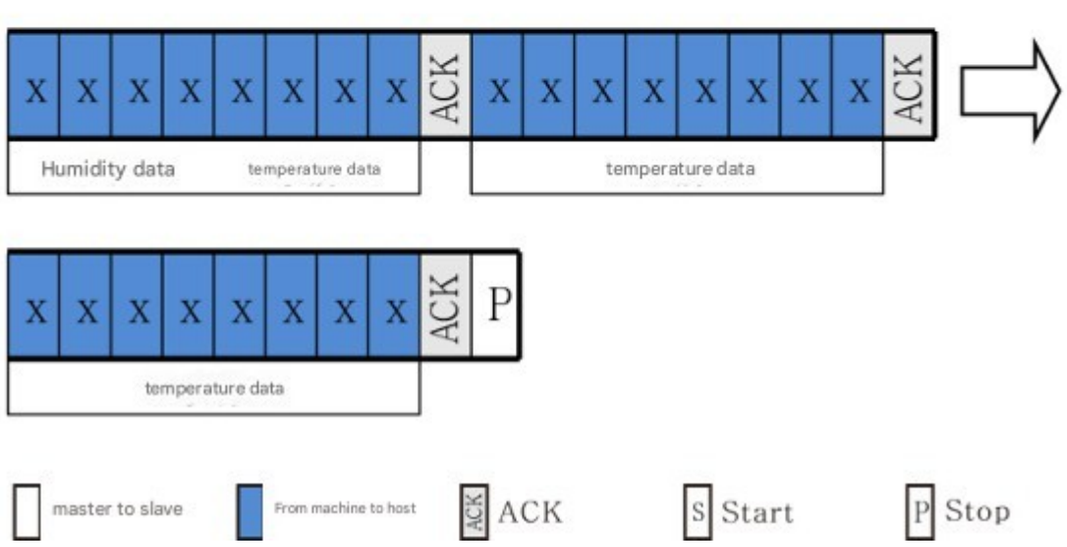
2. Następnie wysyłamy bajt odpowiedzialny za komendę do inicjalizacji czyli: 0xAC
3. Następnie wysyłamy bajt z aktywnym bajtem kalibracji, czyli: 0x08
4. Potem wysyłamy bajt z samymi zerami, czyli: 0x00

Żeby wysłać komendę która ma za zadanie wykonać pomiar na czujniku to musimy wysłać podaną komendę:



Odczyt danych temperatury i wilgotności:





Do odebrania jak widać na powyższym obrazku jest potrzebne dokładnie 6 bajtów, gdzie każdy z bajtów jest opisany na rysunku.

Uwaga: Czujnik potrzebuje czasu na aktywację, a host wysyła polecenie pomiaru (0xAC). Następnie opóźnienie wynosi ponad 75 milisekund i dopiero wtedy odczytywane są przekonwertowane dane i oceniany jest zwrócony kształt. Sprawdzane jest, czy status jest normalny. Jeśli bit statusu [Bit7] ma wartość 0, dane można odczytać normalnie. Gdy czujnik jest zajęty, host musi poczekać na zakończenie przetwarzania danych.

Miękki reset

Miękki reset służy do ponownego uruchomienia czujnika. Po otrzymaniu tego polecenia, system czujnika rozpoczyna ponowną inicjalizację i przywraca domyślne ustawienia. Czas wymagany do miękkiego resetu nie przekracza 20 milisekund.