

# Sprawozdanie z laboratorium 6

HART (WAGO)

Łukasz Janusz  
Marek Generowicz  
09.03.2025



**AGH**

**AGH UNIVERSITY OF KRAKOW**

# 1 Wstęp

Na laboratoriach należało zapoznać się z protokołem HART, zasadami komunikacji oraz praktycznymi aspektami wykorzystania go w przemyśle. W trakcie zajęć przeprowadzono ćwiczenia z wykorzystaniem sterownika *WAGO 750-841* wyposażonym w dwukanałowy analogowy moduł wejścia, który pozwala na komunikację z urządzeniami HART. Elementem pomiarowym natomiast jest *termopara typu K*, która została połączona z modułem WAGO za pomocą przetwornika temperatury *TxIsoRail-HART*.

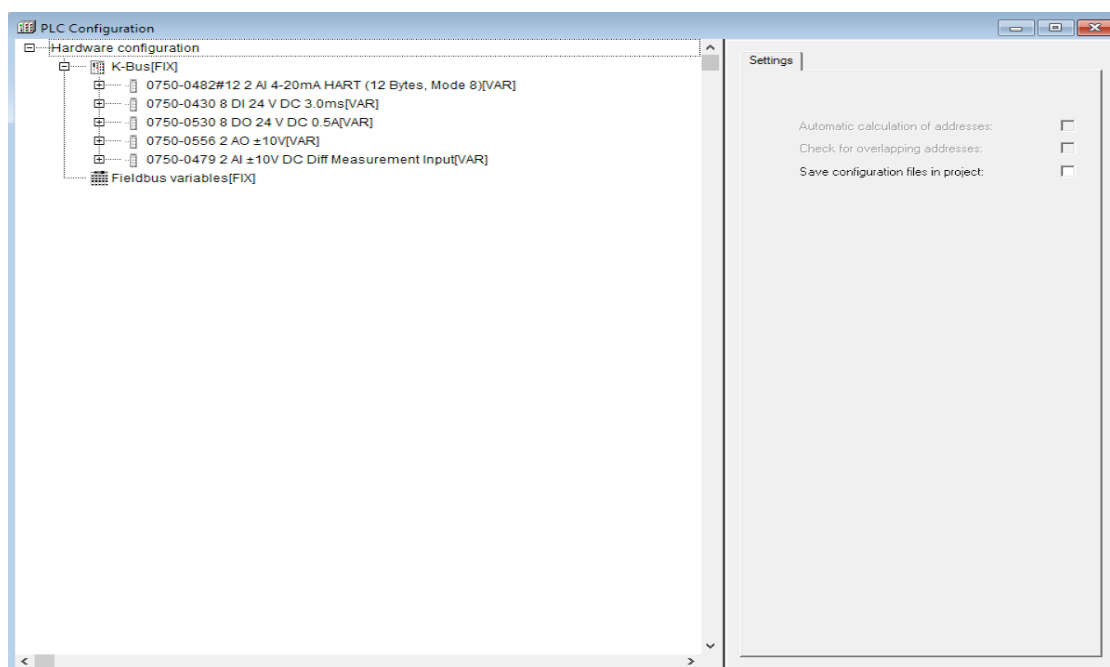
## 1.1 Protokół HART

Protokół HART (*High Addressable Remote Transducer*) jest standardem komunikacyjnym stosowanym w przemyśle, który pozwala na komunikację z urządzeniami pomiarowymi, takimi jak czujniki, przetworniki, zawory, itp. Protokół HART umożliwia przesyłanie danych cyfrowych i analogowych w jednym przewodzie. Komunikacja odbywa się za pomocą sygnałów modulowanych na sygnale prądu stałego, co pozwala na przesyłanie danych cyfrowych wraz z sygnałem analogowym. Protokół HART jest kompatybilny z większością urządzeń pomiarowych, co pozwala na łatwe wdrożenie w istniejących systemach. Urządzenia, które wykorzystują protokół HART, są podzielone na nadrzędne (np. sterowniki PLC) i podrzędne (np. czujniki).

## 2 Przebieg ćwiczenia

### 2.1 Konfiguracja PLC

W pierwszej części zadania należało zaprogramować sterownik *WAGO*. W tym celu należało skorzystać z aplikacji *CoDeSys*. Ważne aby w nowo stworzonym projekcie ustawić *Type od POU* na *Program* a język programowania na *FBD* ze względu na konieczność wykorzystania biblioteki do obsługi *POU* napisanej w tym właśnie języku. Następnie należało dodać moduły wejścia i wyjścia w wirtualnym wnętrzu magistrali. Uzupełniona magistrala wyglądała jak na zdjęcie 1.

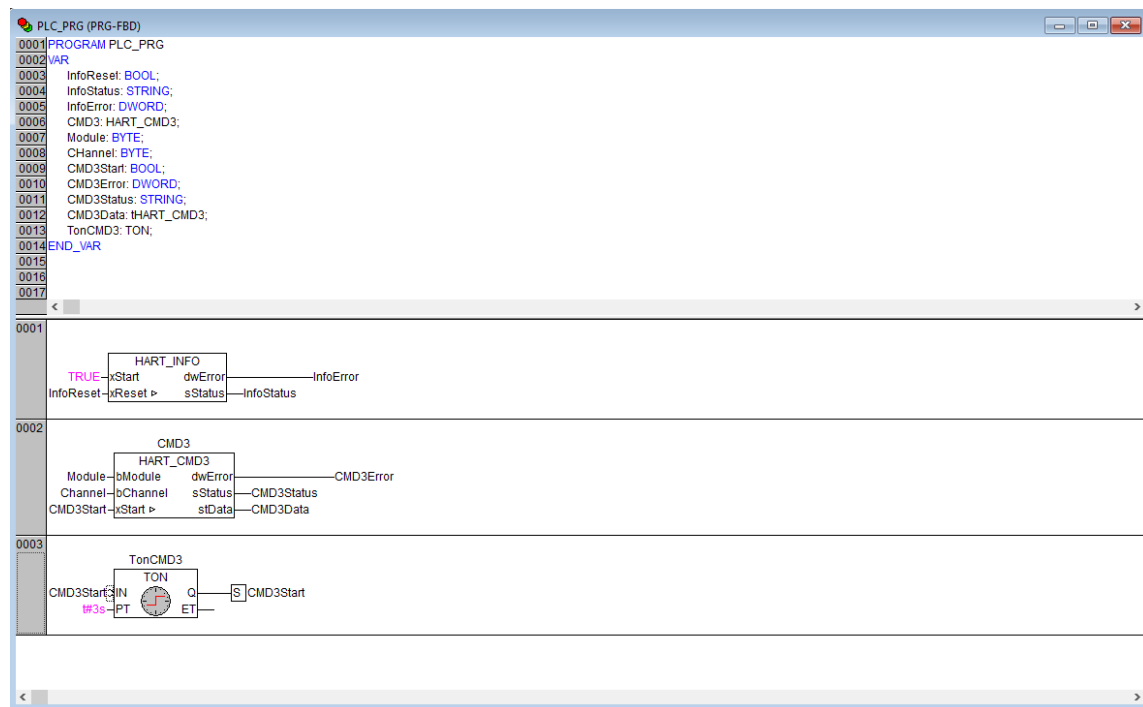


Rysunek 1: Wnętrze magistrali w aplikacji *CoDeSys*.

Przed przystąpieniem do programowania należało skonfigurować parametry komunikacji oraz, w razie gdyby jej nie było, dodać bibliotekę do obsługi komunikacji HART *WagoLibHART\_03.lib*

## 2.2 Program - HART\_INFO

Po skonfigurowaniu PLC należało przejść do napisania kodu obsługującego moduł HART. Okno programu jest podzielone na dwie części, w górnej znajdują się zmienne używane w kodzie, a w dolnej znajduje się logika kodu. Zdjęcie 2 przedstawia całość kodu napisanej na zajęciach aplikacji. Jak widać logika zależy od trzech bloków logicznych.



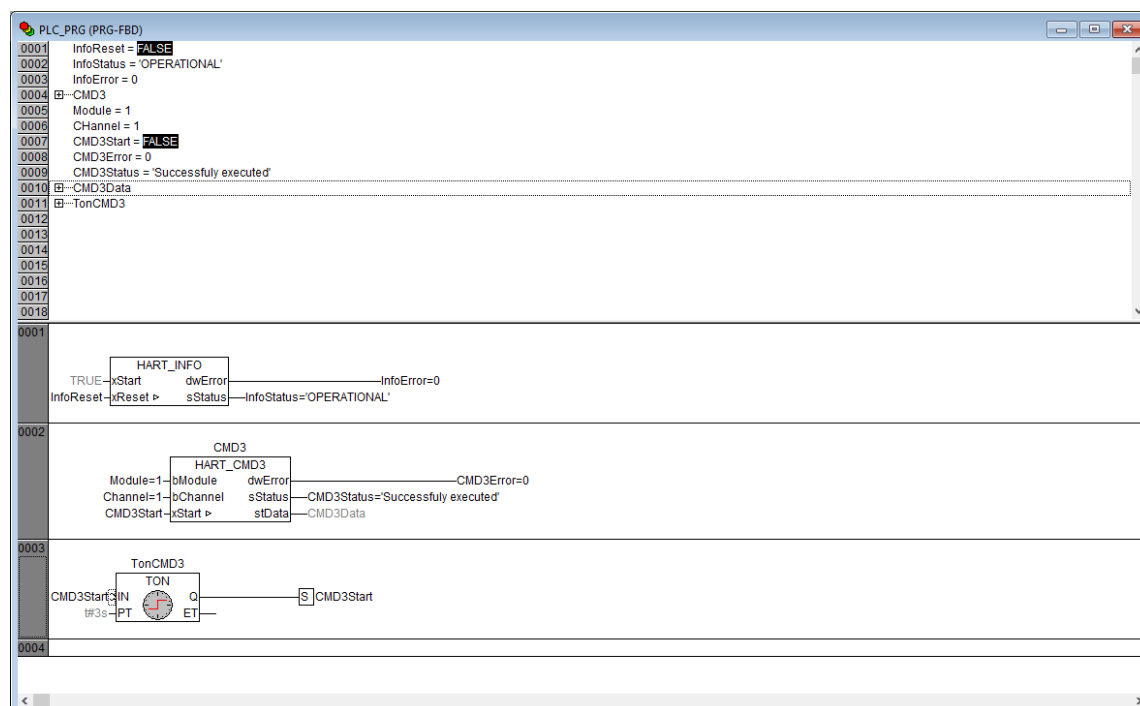
Rysunek 2: Program obsługujący wejście analogowe modułu HART.

Pierwszy z nich *HART\_INFO* służy do odczytu czy cały układ jest w stanie komunikować się z modułem HART.

Drugi blok *HART\_CMD3* za pomocą uniwersalnej komendy HART numer 3 służy do odczytu zmiennych w module HART.

Trzeci blok *TON* jest zegarem, który pozwala na cykliczne wykonywanie logiki drugiego bloku, ponieważ blok ten jest uruchamiany rosnącym zboczem parametru *CMD3Start*. Ważne jest aby wejście timera zanegować aby zegar działał zgodnie z tym co oczekujemy.

Stworzenie takiego programu daje nam możliwość odczytu zmiennych z modułu HART, co jest niezbędne do dalszych działań. W trakcie uruchamiania programu wartości należało pamiętać aby ustawić wartości zmiennych *Module* oraz *Channel* na wartości 1. Wygląd uruchomionego programu przedstawia zdjęcie 3.



Rysunek 3: Uruchomiony program do obsługi modułu HART.

## 2.3 Program - odczyt zmiennych HART

Po poprawnej konfiguracji oraz zaprogramowaniu sterownika, można przystąpić do odczytu zmiennych z modułu HART. W tym celu należy uruchomić program i ustawić poprawne wartości zmiennych *Module* oraz *Channel* na wartości 1. Po uruchomieniu programu, dzięki blokowi *TON* program będzie cyklicznie co 6 sekund aktualizował dane, które moduł HART będzie otrzymywał od termopary typu K.

Aby odczytać wartości odebrane przez termoparę typu K oraz przekazane przez moduł HART, należy w części programu gdzie znajdują się parametry układu rozwinąć część *CMD3Data*. Zdjęcie 4 przedstawia jak wygląda odczyt zmiennych z modułu HART.

0007	CMD3Start = FALSE
0008	CMD3Error = 0
0009	CMD3Status = 'Successfully executed'
0010	[-] CMD3Data
0011	.rVarCurrent = 6.994897
0012	.rVarPrimary = 24.61728
0013	.sUnitSymPrimary = '°C'
0014	.sUnitTxtPrimary = 'Degrees Celsius'
0015	.rVarSecondary = 28.72426
0016	.sUnitSymSecondary = '°C'
0017	.sUnitTxtSecondary = 'Degrees Celsius'
0018	.rVarThird = -8.453844e-002
0019	.sUnitSymThird = 'mV'
0020	.sUnitTxtThird = 'Millivolts'
0021	.rVarFourth = 0
0022	.sUnitSymFourth = ''
0023	.sUnitTxtFourth = ''
0024	[+] TonCMD3

Rysunek 4: Dane otrzymane z modułu HART.

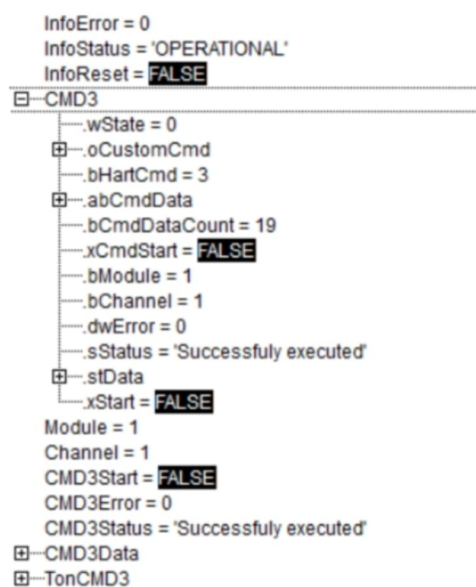
## 2.4 Dane HART

W celu sprawdzenia ilości zmiennych dynamicznych należy skorzystać z tabeli dostępnej w dokumentacji urządzenia.

Dynamic Variables Supported	No. of Response Data Bytes
PV	9
PV, SV	14
PV, SV, TV	19
PV, SV, TV, QV	24

Tabela 1: Zależność między ilością zmiennych dynamicznych a długością tablicy CMD.abCmdData

Aby tabela była użyteczna sprawdzamy wartość pola CMD3.bCmdDataCount, która w naszym przypadku wynosi 19.



Rysunek 5: Wartość pola CMD3.bCmdDataCount

Dzięki temu wiadomo, że nasze urządzenie obsługuje dokładnie 3 zmienne dynamiczne.

Następnie należy sprawdzić jakie informacje są przekazywane w poszczególnych bitach oraz zdekodować i porównać znalezione wartości do danych widniejących w strukturze CMD3DATA. W celu sprawdzenia informacji o poszczególnych polach należy skorzystać z dokumentacji urządzenia.

### Request Data Bytes

Byte	Format	Description
None		

### Response Data Bytes

Byte	Format	Description
0-3	Float	Primary Variable Loop Current (units of milliamps <sup>2</sup> )
4	Enum-8	Primary Variable Units Code (refer to <i>Common Tables Specification</i> )
5-8	Float	Primary Variable
9	Enum-8	Secondary Variable Units Code (refer to <i>Common Tables Specification</i> )
10-13	Float	Secondary Variable
14	Enum-8	Tertiary Variable Units Code (refer to <i>Common Tables Specification</i> )
15-18	Float	Tertiary Variable
19	Enum-8	Quaternary Variable Units Code (refer to <i>Common Tables Specification</i> )
20-23	Float	Quaternary Variable

Tabela 2: Znaczenie poszczególnych bitów w tabeli abCmdData

W celu sprawdzenia wartości *PV* (*Primary Value*) został użyty prosty program w języku C.

```

main.c
1 #include <stdio.h>
2 #include <string.h>
3 int main() {
4     float f;
5     unsigned char b0 = 65; // bajt 0 decymalnie;
6     unsigned char b1 = 196; // bajt 1 decymalnie;
7     unsigned char b2 = 240; // bajt 2 decymalnie;
8     unsigned char b3 = 50; // bajt 3 decymalnie;
9
10    unsigned char b[] = {b3,b2,b1,b0};
11    memcpy(&f,b,sizeof(f));
12    printf("%f",f);
13    return 0;
14 }

Output
24.617283
--- Code Execution Successful ---

0001 InfoReset = FALSE
0002 InfoStatus = 'OPERATIONAL'
0003 InfoError = 0
0004 CB-CMD3
0005 --wState = 0
0006 CB--oCustomCmd
0007 CB--oHwCmd = 3
0008 CB--abCmdData
0009 --abCmdData[0] = 64
0010 --abCmdData[1] = 223
0011 --abCmdData[2] = 214
0012 --abCmdData[3] = 50
0013 --abCmdData[4] = 32
0014 --abCmdData[5] = 65
0015 --abCmdData[6] = 196
0016 --abCmdData[7] = 240
0017 --abCmdData[8] = 50
0018 --abCmdData[9] = 32
  
```

Rysunek 6: Program w języku C do sprawdzenia wartości PV

Jak widać, jego odpowiedź jest zgodna (co do ilości wyświetlanych miejsc po przecinku) z danymi wyświetlanymi w polu CMD3Data.rVarPrimary (Rysunek 4).

Drugim sposobem sprawdzenia wartości *PV* jest internetowy konwerter dostępny pod linkiem <https://www.h-schmidt.net/FloatConverter/IEEE754.html>. Po wprowadzeniu odczytanych wartości otrzymujemy identyczny wynik.



**IEEE 754 Converter, 2024-02**

	Sign	Exponent	Mantissa
<b>Value:</b>	+1	$2^4$	$1 + 0.5385801792144775$
<b>Encoded as:</b>	0	131	4517938
<b>Binary:</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
Decimal Representation	24.617283		
Value actually stored in float:	24.617282867431640625		
Error due to conversion:	0.000000132568359375		
Binary Representation	01000001110001001111000000110010		
Hexadecimal Representation	41c4f032		

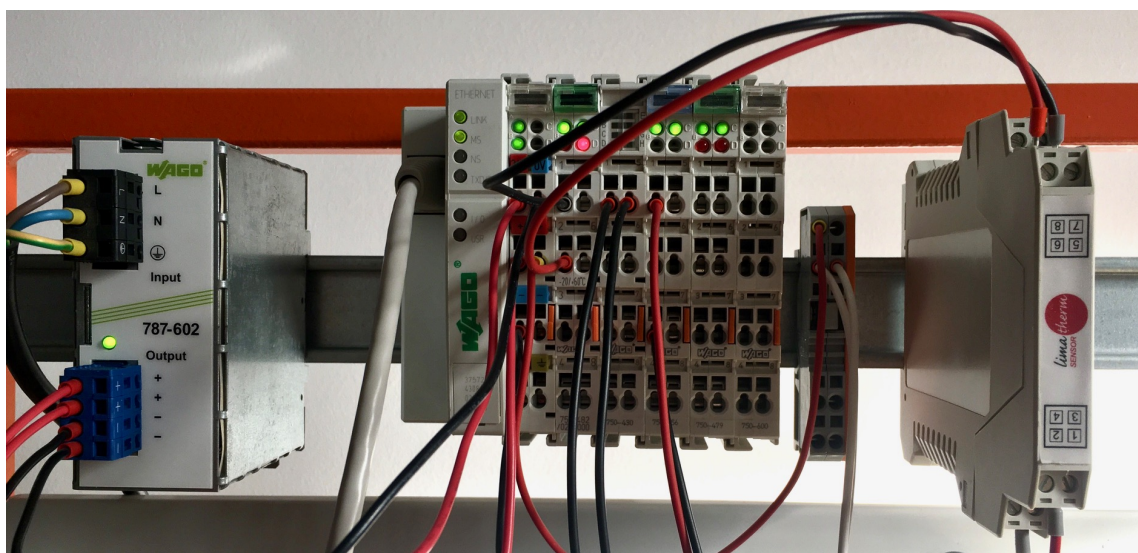
1

-1

Rysunek 7: Konwersja wartości binarnych na wartość PV

### 3 Podsumowanie

Nauka programowania modułu HART jest bardzo przydatna ponieważ pozwala na komunikację z urządzeniami pomiarowymi w przemyśle ponieważ daje możliwość komunikacji dwukierunkowej nawet w niesprzyjających warunkach. W trakcie laboratoriów udało nam się zaprogramować sterownik *WAGO 750-841*, odczytać dane z modułu HART oraz nauczyć się odczytywać "surowe" dane aby odczytać poszukiwane informacje. Dzięki temu zdobyliśmy praktyczną wiedzę na temat protokołu HART co umożliwi nam w przyszłości praktyczne wykorzystanie tej wiedzy w przemyśle.



Rysunek 8: Wygląd modułu HART