PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

Ośrodek Automatyzacji Procesów Produkcji

440 A

Główny wykonawca

Wykonawcy dr inż. Marian Wrzesień
mgr inż. Jacek Domaj

Konsultant

Nr zlecenia 9532

"Wykonanie oprogramowania współpracy układu sterowania IRp z inteligentnymi układami sensorycznymi w tym oprogramowanie współpracy z czujnikami wizyjnymi 1D i 2D.
Etap 1: Wykonanie i uruchomienie oprogramowania dla pakietu MIO6 realizującego protokół przesyłu informacji po 4-kanałowym złączu szeregowym pomiędzy czujnikiem wizyjnym a jednostką centralną układu sterowania robota IRp-6. Przeprowadzenie badań funkcjonalnych z czujnikiem wizyjnym 1D."

Zleceniodawca

Praca własna PIAP.

Pracę rozpoczęto dnia 1988.01.01 zakończono dnia 1990.02.28

Kierownik Ośrodka
dr inż. Marian Wrzesień

Z-ca Dyrektora d/s Automatyki i Pomiarów
doc. dr inż. Tadeusz Gałązka

Praca zawiera:

stron

rysunków

fotografii

tabel

tablic

załączników 1

Nr rejestr. 6423

Rozdzielnik - ilość egz:

Egz. 1 ARCHIWUM

Egz. 2 OAE

Egz. 3 OAP

Egz. 4

Egz. 5

Egz. 6

Analiza deskryptorowa

~~CZUJNIKI WIZYJNE, ZŁĄCZE SZEREGOWE WIELOKANAŁOWE~~ /

UKŁADY STEROWANIA, OPROGRAMOWANIE, CZUJNIK

Analiza dokumentacyjna

Przedstawiono handler dla pakietu MI-06 oraz oprogramowanie dla procedury przesyłu informacji wg protokołu przesyłu dla 4-kanałowego złącza szeregowego oraz badania funkcjonalne czujnika wizyjnego TD.

Tytuły poprzednich sprawozdań

621.397.6 Urządzenia wizyjne
681.3.06 Oprogramowanie

UKD

PIAP 41/88 10000

2

SPIS TREŚCI:

1. Omówienie prac wykonanych w ramach etapu 1 zlecenia 9532.

Celem prac było opracowanie oprogramowania umożliwiającego komunikowanie się robota z czujnikiem wizyjnym. Oprogramowanie czujnika wizyjnego było prowadzone w ramach tematu RP-63, natomiast oprogramowanie zapewniające łączność pomiędzy układem sterowania robota, a czujnikiem realizowane jest w ramach pracy własnej 9532. Praca ta wspomaga szersze działania zmierzające do opracowania funkcji adaptacyjnych w programie sterującym robota (zlecenie 1122), których elementem jest realizacja oprogramowania inteligentnych układów sensorycznych.

Z przyjętego planu pstępowania wyniknął zakres pracy własnej 9532, w której opracowano:

- pięciokanałowy pakiet komunikacyjny na bazie modułu MI-06,
- zbiór plików stanowiących pakiet obsługujący komunikację od strony robota,
- przeprowadzenie badań funkcjonalnych czujnika 1D we współpracy z układem sterowania robota.

Poniżej przedstawiono wyniki prac etapu 1 zlecenia, z których dwa pierwsze opracowania stanowią załączniki ujmujące zamkniętą całość przeznaczoną do zastosowania w dalszych pracach PIAP.

Przemysłowy Instytut Automatyki i Pomiarów

Warszawa Aleje Jerozolimskie 202

PIĘCIOKANAŁOWY PAKIET KOMUNIKACYJNY DLA POTRZEB ROBOTOW IRP-6/60

(na bazie modułu MI06)

mgr inż. Jacek Dunaj

Marzec 1990

SPIS TRESCI:

1. SRODOWISKO SPRZETOWE

1.1. Informacje dotyczące pakietu MI06

Pakiet komunikacyjny został zrealizowany na bazie modułu MI06. Szczegółowe informacje dotyczące tej jednostki zawarto w DTR "Moduł transmisji szeregowej MI06". Tutaj zostaną podane jedynie te istotne z punktu widzenia oprogramowania i wykorzystania tego pakietu.

Pakiet MI06 zawiera m.in. sześć programowalnych układów transmisji szeregowej USART 8251A, z których pięć pierwszych moze pracować w trybie asynchronicznym a szósty w trybach: synchronicznym i asynchronicznym. Układy te można testować programowo odczytując słowo stanu każdego z nich, można je też wykorzystać do generowania przerwań na wejściach dwóch programowalnych sterowników przerwań 8259A. Osobny blok stanowią cztery dzielniki programowe (timery) 8253A, które wytwarzają sygnały o częstotliwościach określających prędkosci transmisji układów USART 8251A, niezależnie dla kierunków nadawania i odbioru. Przyporządkowanie wyjść układów 8253A wejściom poszczególnych układów USART podano w poniższej tabeli:

Dzielnik programowany		USART	
układ:	wyjście:	wejście	układ:
CLK 0	OUT 0	T x C	USART 1
	OUT 1	R x C	
	OUT 2	T x C	USART 2
CLK 1	OUT 0	R x C	
	OUT 1	T x C	USART 3
	OUT 2	R x C	
CLK 2	OUT 0	T x C	USART 4
	OUT 1	R x C	
	OUT 2	T x C	USART 5
CLK 3	OUT 0	R x C	
	OUT 1	T x C	USART 6
	OUT 2	R x C	

Układ kontrolera przerwań zrealizowano na bazie dwóch programowalnych układów PIC 8259A. Obsługuje on przerwania od kanałow zewnętrznych, informując system o wysłaniu lub przyjęciu znaku w każdym z kanałow. Umożliwia on priorytetową, wektoryzowaną obsługę przerwań generowanych przez poszczególne kanały transmisji szeregowej w momencie wysłania lub odebrania znaku (TxRdy, RxRdy), a także przy wykryciu sekwencji znaków synchronizujących (przy transmisji synchronicznejw kanale 6). Układy przewidziane są do pracy buforowanej w trybie SLAVE ustawionym podczas inicjalizacji. Przyporządkowanie linii przerwań określono w poniższej tabeli.

Zródło przerwania:	Nazwa sygnału:	Oznaczenie pinu danego układu 8259A:	Kontroler przerwań:
USART 1	TxRdy RxRdy	IR0 IR1	PIC 0
USART 2	TxRdy RxRdy	IR2 IR3	
USART 3	TxRdy RxRdy	IR4 IR5	
USART 4	TxRdy RxRdy	IR6 IR7	
USART 5	TxRdy RxRdy	IR0 IR1	PIC 1
USART 6	TxRdy RxRdy Syndet	IR2 IR3 IR4	
CLK 3 *)	OUT 2	IR5	
-	-	IR6	
-	-	IR7	

*) Wyjście OUT 2 z dzielnika CLK 3 można wykorzystać jako źrodło przerwania w przypadku jednakowych prędkości transmisji nadawanie - odbiór USART-a nr 6. Należy wówczas dokonać odpowiednich zmian na polu krosowym J-5.

Sygnały wyjściowe INT z programowalnych sterowników przerwań są dołączone do systemowego sterownika przerwań (MASTER) znajdującego się na pakiecie MM16. Przyporządkowanie wyjść ze sterowników SLAVE poszczególnym wejściom układu MASTER uzyskuje się na polu krosowym J-1 modułu MI06.

Pojedyńczy pakiet MI06 zajmuje w przetrzeni adresowej portów systemu ciągły obszar 128-elementowego bloku. Najstarszy bajt adresu każdego elementu pakietu jest ustalany za pomocą jego pola krosowego J-2, a kolejne osiem bitów wyznacza układ dekodera adresu na podstawie zawartości programowalnej pamięci PROM TTL 24S10. Ostatnie siedem bitow 16-bajtowego adresu każdego układu pakietu jest przyporządkowane na stałe danemu układowi, zgodnie z poniższą tabelą:

Układ:	Linie adresowe:						
	A6:	A5:	A4:	A3:	A2:	A1:	A0:
USART 1	0	0	0	0	*	-	-
USART 2	0	0	0	1	*	-	-
USART 3	0	0	1	0	*	-	-
USART 4	0	0	1	1	*	-	-
USART 5	0	1	0	0	*	-	-
USART 6	0	1	0	1	*	-	-
Niewykorzystywany	0	1	1	0	-	-	-
Niewykorzystywany	0	1	1	1	-	-	-
CLK 0	1	0	0	0	*	*	-
CLK 1	1	0	0	1	*	*	-
CLK 2	1	0	1	0	*	*	-
CLK 3	1	0	1	1	*	*	-
PB 0	1	1	0	0	-	-	-
PB 1	1	1	0	1	-	-	-
PIC 0	1	1	1	0	*	-	-
PIC 1	1	1	1	1	*	-	-

- bity niedekodowane,
* bity wykorzystywane do adresowania rejestrów wewnętrznych danego układu.

1.2. Informacje dotyczące ustaleń sprzętowych na pakiecie MI06

Ponieważ pakiet komunikacyjny zrealizowany na bazie modułu MI06 jest projektowany do współpracy:

STEROWNIK ROBOTA <-----> PAKIET WIZYJNY

zaszła konieczność dopasowania przestrzeni adresowej portów modułu do możliwości adresowych szafy sterowniczej robota. Ze względów oszczędnościowych poszczególne pakiety szafy sterowniczej robota posiadają tak wykonany układ dekodujący ich adresy, że reaguje on tylko na młodszy bajt 2-bajtowego adresu portu (w programach sterujących robota określany jest cały 16-bitowy adres). Oznacza to, że przestrzeń adresowa portów systemu sterownika robota jest ograniczona tylko do 256 elementów, z których trzeba wyodrębnić ciągły obszar 128-elementowy dla adresacji poszczególnych układów pakietu MI06. Stało się to wykonalne po zmianie młodszych bajtów adresów poszczególnych układów pakietu MI50 (sterownik pamięci kasetowej) z wartości F0H, F2H, F4H, F6H, F8H na odpowiednio 70H, 72H, 74H, 76H i 78H. Przyjęto zatem, że pełnym adresem lokalizującym początek przestrzeni adresowej pakietu MI06 jest 8080H, co implikuje następujące przyporządkowanie adresów poszczególnych układów tego pakietu:

Układ:	Adres hexadecymalny:	Adresacja:
USART 1:	80 80 80 84	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
USART 2:	80 88 80 8C	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
USART 3:	80 90 80 94	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
USART 4:	80 98 80 9C	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
USART 5:	80 A0 80 A4	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
USART 6:	80 A8 80 AC	IN: 8251_DATA->DATA_BUS, OUT: DATA_BUS->8251_DATA IN: STATUS->DATA_BUS, OUT: DATA_BUS->CONTROL
Niew. 1:	80 B0	
Niew. 2:	80 B8	
CLK 0 (timer 8253A):	80 C0 80 C2 80 C4 80 C6	Licznik 0 Licznik 1 Licznik 2 Rejestr słowa sterowania
CLK 1 (timer 8253A):	80 C8 80 CA 80 CC 80 CE	Licznik 0 Licznik 1 Licznik 2 Rejestr słowa sterowania
CLK 2 (timer 8253A):	80 D0 80 D2 80 D4 80 D7	Licznik 0 Licznik 1 Licznik 2 Rejestr słowa sterowania
CLK 3 (timer 8253A):	80 D8 80 DA 80 DC 80 DE	Licznik 0 Licznik 1 Licznik 2 Rejestr słowa sterowania
PB 0:	80 E0	
PB 1:	80 E8	
PIC 0 (8259A):	80 F0 80 F4	ICW1 OCW2 OCW3 ICW2 ICW3 ICW4 OCW1
PIC 1 (8259A):	80 F8 80 FC	ICW1 OCW2 OCW33 ICW2 ICW3 ICW4 OCW1

Z pozostałych ustaleń sprzętowych, istotnych dla oprogramowania przyjęto:

- wybór wstępnego podziału częstotliwości zegara dla układów USART: (pole krosowe J-4) - podział przez 2,

- przyporządkowanie wyjść sterowników przerwań:
  SLAVE_PIC_0 --> wejście IR0 układu MASTER_PIC na pakiecie MM16,
  SLAVE_PIC_1 --> wejście IR1 układu MASTER_PIC na pakiecie MM16.

2. OPROGRAMOWANIE PAKIETU KOMUNIKACYJNEGO

U W A G A !

W poniższym tekście wszelkie oznaczenia stałych, zmiennych i funkcji mają postać taką jak w tekście źródlowym, napisanym w asemblerze ASM86. Przy wykorzystaniu ich z poziomu języka C, pod kompilatorem MWC86, nalezy być świadomym, że dostępne są w nim tylko te oznaczenia, które w tekście źródłowym zdefiniowano jako PUBLIC i ich nazwy w definicji zakończono znakiem "_" (inderscore). Przy stosownych odwołaniach, wykonywanych w języku C znak ten jest pomijany (nie jest pomijany w odwołaniach asemblerowych). Zasada ta obowiązuje także w odwrotnym kierunku: przy korzystniu na poziomie asemblera ASM86 z wielkości i funkcji zdefiniowanych w jezyku C. Blizsze informacje na temat zasad łaczenia podprogramów napisanych w asemblerze i języku C, a także wzajemnej wymiany parametrów pomiedzy tymi programami można znaleźć w rozdziale pt. "Using the Assembler" podręcznika "MWC 86™ C Compiler Programer's Reference Manual"

Pakiet MI06 zapewnia sprzętową możliwość równoczesnego podłączenia do układu sterowania robotem sześciu urządzeń zewnętrznych. W obecnej wersji handler komunikacyjny pozwala na współpracę od strony programowej pięciu takich urządzeń, a wymiana informacji następuje poprzez asynchroniczne porty transmisji szeregowej USART 8251A.

W celu zapewnienia wymiany informacji za pośrednictwem pakietu MI06 każdemu kanałowi transmisji przyporządkowano w pamięci operacyjnej dwa bufory:

- bufor wejściowy, oznaczany w tekście źródłowym programu jako **mi06_input_buffer_n**, gdzie n ∈ ⟨1, 5⟩
- bufor wyjściowy, oznaczany w tekście źródłowym programu jako **mi06_output_buffer_n**, gdzie n ∈ ⟨1, 5⟩

2.1. Bufor wejściowy

Kolejne znaki odbierane w danym kanale zostają wpisane bezpośrednio, bez interpretacji ich znaczenia do "SIZE_OF_INPUT_BUFFER_n_" wymiarowego (n ∈ ⟨1,5⟩) bufora wejściowego. Każdemu elementowi tego bufora odpowiada pojedyńcza komórka pamięci RAM. Bufor wejściowy jest buforem cyklicznym, tzn. że odbierane znaki są wstawiane do niego sekwencyjnie w kierunku rosnących adresów. Dojście do końca bufora powoduje automatyczne ustawienie indeksu sterującego jego wypełnianiem na początek.

Wypełnianiem i odczytem informacji z bufora wejściowego sterują następujące dwa indeksy:

1. **Indeks odczytu z bufora wejściowego**, będący numerem komórki pamięci, liczonym względem początku bufora, zawierającej ostatni odczytany znak z tego bufora. Indeks ten jest zwiększany o jeden bezpośrednio przed odczytaniem znaku z bufora, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako "**indeks_read_of_in_buffer_n**", gdzie n jest oznaczeniem odpowiedniego kanału transmisji.

2. **Indeks zapisu do bufora wejściowego**, będący numerem komórki pamięci, liczonym względem początku bufora, zawierajacej ostatni, odebrany znak z danego kanału transmisji szeregowej. Indeks ten jest zwiększany o jeden bezpośrednio przed wpisaniem do bufora wejściowego odebranego znaku, a po dojściu do końca tego bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako "indeks_input_of_in_buffer_n", gdzie n jest oznaczeniem odpowiedniego kanału transmisji.

Oprócz obu tych indeksów do obsługi bufora wejściowego wykorzystano dodatkowo **licznik obsługi bufora wejściowego**, który:

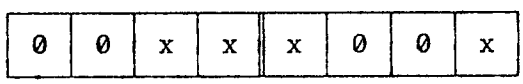
- jest zwiększany o jeden bezpośrednio przed wpisaniem do bufora wejściowego odebranego znaku,
- jest zmniejszany o jeden bezpośrednio po odczytaniu znaku z bufora wejściowego.

Licznik obsługi n-tego bufora wejściowego, oznaczany w tekstach źródłowych jako "in_counter_n_" jest wykorzystywany do:

- określania momentu zakończenia czynności odczytu znaków z tego bufora (jeśli wartość licznika wynosi zero),
- określenia sytuacji awaryjnej, w której bufor wejściowy zostaje przepełniony (wartość wpisana do licznika pozostaje wtedy równa rozmiarowi bufora wejściowego, a stan awaryjny jest sygnalizowany procedurze nadrzędnej poprzez ustawienie odpowiedniego wskaźnika błędu.

Zmienne in_counter_n_ n ∈ <1,5> są dwubajtowymi zmiennymi typu PUBLIC i mogą być wykorzystywane jako zmienne "READ-ONLY" w podprogramach użytkownika.

Oprócz informacji zawartych w buforze wejściowym, indeksach jego zapisu i odczytu oraz w liczniku obsługi bufora wejściowego, procedura realizującą transmisję pomiędzy układem sensorycznym a sterownikiem robota ustawia następujące 1-bitowe wskaźniki błędów w 1-bajtowym słowie błędów:



1 - błąd ramki,
0 - brak błędu ramki.

1 - błąd przepełnienia,
0 - brak błędu przepełnienia.

1 - przepełnienie bufora wejściowego,
0 - bufor wejściowy nie jest przepełniony.

1 - błąd parzystości,
0 - brak błędu parzystości.

Błędy ramki, przepełnienia i parzystości są przepisywane ze słowa stanu układu USART 8251A obsługującego dany kanał transmisji. Informacja o błędach jest uzupełniana po odebraniu kolejnego znaku z danego kanału transmisji (wykonywana jest suma logiczna informacji archiwalnej i bieżącej), może być w kazdej chwili odczytana przez użytkownika i tylko przez niego jest kasowana. Stosowne oznaczenia to:

read_error_indicator_n_ - procedura (typu PUBLIC) odczytująca wskaźniki błędów z 1-bajtowego słowa błedów,

reset_error_n_ - procedura (typu PUBLIC) kasująca wszystkie wskaźniki błędów w 1-bajtowym słowie błędów.

Wywołanie tych procedur ma następującą postać:

- z poziomu asemblera ASM86:

```
CALL read_error_indicator_n_  n ∈ <1,5>
CALL reset_error_n_           n ∈ <1,5>
```

- z poziomu języka C:

```
read_error_indicator_n ()     n ∈ <1,5>
reset_error_n ()              n ∈ <1,5>
```

W przypadku procedury read_error_indicator_n_ wskaźniki błedów są zwracane do podprogramu wywołującego poprzez rejestr AL (AH=00H). W przypadku podprogramu nadrzędnego napisanego w języku C dla kompilatora MWC86 oznacza to, że zwrot następuje poprzez nazwę funkcji.

U W A G A !

Użytkownik może w swoich podprogramach wykorzystywać stała SIZE_OF_INPUT_BUFFER_n_ (n ∈ <1,5>, określającą rozmiar n-tego bufora wejściowego, gdyż jest ona typu PUBLIC. Należy jednak zwrócic uwagę na sposób jej deklaracji w procedurach nadrzędnych:

- w przypadku programu nadrzędnego napisanego w języku asemblera ASM86 stosowna deklaracja musi mieć postać:

```
EXTRN SIZE_OF_INPUT_BUFFER_n_: ABS
```

a przykładowe odwołanie:

```
MOV AX, SIZE_OF_INPUT_BUFFER_1_
```

- w przypadku programu nadrzędnego napisanego w języku C dla kompilatora MWC86 stosowna deklaracja musi mieć postać:

```
extern unsigned int SIZE_OF_INPUT_BUFFER_n;
```

a przykładowe odwołanie:

```
c = &SIZE_OF_INPUT_BUFFER_1;
```

2.2. Bufor wyjściowy

Użytkownik wpisuje przesyłkę w postaci ciągu kolejnych znaków stanowiących informację dla układu zewnętrznego do "SIZE_OF_OUTPUT_BUFFER_n" wymiarowego bufora wyjściowego danego kanału transmisji. Wpisywana informacja ma już postać zakodowaną zgodnie z formatem przesyłek wymaganym przez dany układ zewnętrzny. Każdemu elementowi tego bufora odpowiada pojedyńcza komórka pamięci RAM. Bufor wyjściowy jest buforem cyklicznym, tzn. że znaki są wpisywane do niego sekwencyjnie w kierunku rosnących adresów. Dojście do końca bufora powoduje automatyczne ustawienie indeksu sterującego jego wypełnianiem na początek tego bufora.

Wypełnianiem i wysyłaniem znaków z bufora wyjściowego sterują dwa następujące indeksy:

1. **Indeks zapisu do bufora wyjściowego**, będący numerem komórki pamięci, liczonym od początku bufora, zawierającej ostatni zapisany znak do tego bufora. Indeks ten jest zwiększany o jeden bezpośrednio przed wstawieniem znaku do bufora, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako "indeks_write_of_out_buffer_n", gdzie n jest oznaczeniem odpowiedniego kanału transmisji.

2. **Indeks wysłanych znaków z bufora wyjściowego**, będący numerem komórki pamięci, liczonym od początku bufora, z której wysłano ostatni znak. Indeks ten jest zwiększany o jeden bezpośrednio przed wysłaniem znaku z bufora wyjściowego do odpowiedniego urządzenia zewnętrznego, po dojściu do końca tego bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako "indeks_trans_of_out_buffer_n", gdzie n jest oznaczeniem odpowiedniego kanału transmisji.

Oprócz obu tych indeksów, do obsługi bufora wyjściowego wykorzystano dodatkowo licznik obsługi bufora wyjściowego, oznaczany jako "out_counter_n", który:

- jest zmniejszany o jeden bezpośrednio po wysłaniu znaku z bufora wyjściowego do układu zewnętrznego,

- jest zwiększany o jeden bezpośrednio przed wpisaniem znaku do bufora wyjściowego.

Licznik obsługi bufora wyjściowego spełnia taką samą role jak jego odpowiednik dla bufora wejściowego. Zmienne out_counter_n_ n ∈ ⟨1,5⟩ są dwubajtowymi zmiennymi typu PUBLIC i mogą być wykorzystywane jako zmienne "READ-ONLY" w podprogramach użytkownika.

U W A G A !

Użytkownik może w swoich podprogramach wykorzystywać stała SIZE_OF_OUTPUT_BUFFER_n_ (n ∈ ⟨1,5⟩, określającą rozmiar n-tego bufora wyjściowego, gdyż jest ona typu PUBLIC. Należy jednak zwrócic uwagę na sposób jej deklaracji w procedurach nadrzędnych:

- w przypadku programu nadrzędnego napisanego w języku asemblera ASM86 stosowna deklaracja musi mieć postać:

```
EXTRN SIZE_OF_OUTPUT_BUFFER_n_: ABS
```

a przykładowe odwołanie:

```
MOV AX, SIZE_OF_OUTPUT_BUFFER_1_
```

- w przypadku programu nadrzędnego napisanego w języku C dla kompilatora MWC86 stosowna deklaracja musi mieć postać:

```
extern unsigned int SIZE_OF_OUTPUT_BUFFER_n;
```

a przykładowe odwołanie:

```
c = &SIZE_OF_OUTPUT_BUFFER_1;
```

2.3. Realizacja transmisji przez użytkownika

Od strony użytkowej transmisja z/do urządzeń zewnętrznych poprzez pakiet MI06 sprowadza się tylko do odczytu/wpisania danych do odpowiednich buforów i śledzenia stosownej informacji o sytuacjach awaryjnych (przepełnienie buforów, błędy wynikające z samej transmisji). Sama wysyłka danych z bufora wyjściowego i wpisywanie danych do bufora wejściowego, wraz z modyfikacją związanych z tym zmiennych, odbywa się automatycznie, poza kontrolą użytkownika, z wykorzystaniem podprogramów obsługi przerwań generowanych przez układy USART 8251A pakietu MI06.

2.3.1. Wysyłanie danych do urządzenia zewnętrznego

Użytkownik chcac wysłać pojedyńczy znak do urządzenia zewnętrzego, podłaczonego do n-tego kanału transmisji szeregowej (n ∈ <1,5>) wpisuje ten znak do n-tego bufora wyjściowego za pośrednictwem procedury "write_to_output_buffer_n_" (typu PUBLIC). Wpisywany znak jest parametrem wejściowym, a zatem:

- w przypadku wywołania procedury z poziomu języka C występuje jako jedyny parametr wywołania i może mieć typ "unsigned int" lub "unsigned char":

  write_to_output_buffer_n (parametr) n ∈ <1, 5>

- w przypadku wywołania z poziomu asemblera parametr zostaje złożony na stosie, jako młodszy bajt składanego słowa, bezpośrednio przed wywołaniem procedury:

  CALL write_to_output_buffer_n_ n ∈ <1, 5>

Wywołanie procedury "write_to_output_buffer_n_" pociąga za sobą wpisanie znaku do odpowiedniego bufora wyjściowego i, w przypadku gdy układ USART 8259A sygnalizuje pusty bufor nadawczy (bit TxRDY slowa stanu tego USART-a jest równy 1), programowe zainicjowanie odpowiedniej procedury obsługi przerwania (procedury o nazwiach TxRDY_n, gdzie n ∈ <1,5>). Kolejne przerwania są już generowane sprzętowo sygnałem TxRDY danego USART-a i pojawiają się do chwili opróżnienia bufora wyjściowego (zliczenia przez licznik obsługi bufora wyjściowego do zera).

Procedura "write_to_output_buffer_n_" zwraca do programu nadrzędnego wartość parametru, z którym została wywołana. W przypadku, gdy bufor wyjściowy nie zawierał miejsca na wpisanie tego znaku (znak ten zamazałby inny, jeszcze niewysłany znak) wpisanie aktualnego znaku nie jest wykonane, wszelkie zmienne sterujące wypełnianiem bufora wyjściowego pozostają bez zmian, a do programu nadrzędnego zwracana jest wartość 0000H. Zwrot informacji do programu nadrzędnego następuje poprzez rejestr AX (dla programu nadrzędnego napisanego w asemblerze ASM86) lub poprzez nazwe funkcji (dla programu nadrzędnego napisanego w języku C dla kompilatora MWC86).

U W A G A !

Po wywolaniu procedury "write_to_output_buffer_n_" z programu napisanego w asemblerze ASM86 należy po powrocie do tego programu zwiększyc wskaźnik stosu o 2. Wynika to stąd, że procedura ta była pisana z myślą o wywoływaniu jej z programów napisanych w języku C dla kompilatora MWC86, który "automatycznie" modyfikuje wskaźnik stosu o taką wartość, jaka

wynika z ilości parametrów, z którą wywołano daną procedurę (wywolanie na poziomie języka C).

2.3.2. Odbiór informacji z urządzenia zewnętrznego

Odbiór informacji z urządzenia zewnętrznego i wpisanie jej do bufora wejściowego obslugującego n-ty kanał transmisji (n ∈ <1,5>) wykonuje procedura o nazwie **RxRDY_n**. Jest to procedura obsługi przerwania generowanego sygnałem RxRDY danego układu USART 8251A. Oprócz wpisania do bufora odebranego znaku i modyfikacji zmiennych sterujących wypełnianiem tego bufora procedura ta ustawia także odpowiednie wskaźniki bledów transmisji i przepełnienia bufora wejściowego w 1-bajtowym słowie błedów.

Od strony użytkowej odczyt pojedyńczego znaku z n-tego bufora wejściowego z równoczesną modyfikacją parametrów sterujących tym odczytem wykonuje procedura "**read_from_input_buffer_1_**" (typu PUBLIC). Procedura zwraca do programu nadrzędnego wartość odczytanego bajtu, a w przypadku, gdy bufor wejściowy nie zawierał żadnego, jeszcze nieodczytanego znaku, zwracana jest wartość 0000H. Zwrot informacji do programu nadrzędnego wykonywany jest poprzez rejestr AX (dla programu nadrzędnego napisanego w asemblerze ASM86) lub poprzez nazwę funkcji (dla programu nadrzędnego napisanego w jezyku C dla kompilatora MWC86).

Oprócz informacji dostępnych w buforach wejściowych użytkownik ma dostęp do 1-bajtowego słowa błędów o nazwie error_indicator_n_ (zmienna typu PUBLIC, jej opis zamieszczono wcześniej). Kasowanie wskaźników błędów dla n-tego kanału wykonuje procedura "**reset_error_n_**" (typu PUBLIC).

2.4. Dołączanie oprogramowania pakietu komunikacyjnego do programu użytkowego

Oprogramowanie pakietu komunikacyjnego można podzielić na dwie części: część dotyczącą inicjowania samego sprzętu, tzn. poszczególnych układów modułu MI06 oraz systemowego sterownika przerwań PIC 8259A modułu MM16, oraz część dotyczącą funkcji wykonywanych przez kolejne kanały transmisji szeregowej.

Część pierwsza obejmuje następujące pliki:

1. Plik MI06_HAR.A86, zawierający wyrażenia ustalające adresy poszczególnych układów modułu MI06 oraz adresy wyznaczające dostęp do systemowego układu PIC 8259A na pakiecie MM16.

2. Plik MI06_51A.A86, zawierający podprogram mi06_51a_ (typu PUBLIC), który ustawia parametry transmisji sześciu układów USART 8251A pakietu MI06.

3. Plik MI06_53A, zawierający podprogram mi06_53a_ (typu PUBLIC), który ustawia tryb pracy czterech programowalnych dzielników 8253A. Pozwalają one ustalić szybkości transmisji, z którymi pracują poszególne porty transmisji szeregowej 8251A modułu MI06 w obu lierunkach: nadawanie/odbiór.

4. Plik MI06_59A.A86, zawierający podprogram mi06_59a_ (typu PUBLIC), ktory inicjuje programowale sterowniki przerwań PIC 8259A pracujące w reżimie SLAVE na pakiecie MI06 i w reżimie MASTER na pakiecie MM16.

5. Plik MI06_INI.A86, zawierający podprogram mi06_ini_ (typu PUBLIC), który wywołuje procedury opisane w punktach 2 ÷ 4, a także w zależności od potrzeb, inicjuje parametry obsługi programowej poszczególnych kanałów transmisji (podprogramy kanal_n_, gdzie n $\in$ ⟨1, 5⟩ z pliku KANAL_n.A86). Procedura ta ustawia także wektory przerwań do podprogramow obsługi przypadkowych przerwań, które mogą się pojawić na poszczególnych wejściach układów SLAVE_PIC_0 i SLAVE_PIC_1 modułu MI06.

Procedura mi06_ini_ jest tym podprogramem, który musi wywołać użytkownik aby móc korzystać z oprogramowania pakietu komunikacyjnego za pomocą modułu MI06.

Część druga oprogramowania pakietu komunikacyjnego ma budowę modułową, tzn. w zależności od potrzeb mogą być dołączane procedury obsługi kolejnych kanałów transmisji szeregowej. Komplet takich procedur dla n-tego kanału zawiera plik o nazwie "KANAL_n.A86". Są to następujące podprogramy:

- podprogram KANAL_n, (typu PUBLIC) inicjujący zmienne związane z obsługą buforów n-tego kanalu transmisji szeregowej, a także inicjujący odpowiedni wektor przerwań do procedur związanych z odbieraniem/wysyłką znakow z/do tego kanału,

- podprogram TxRDY_n obsługi przerwania od sygnału TxRDY USART-a nr n,

- podprogram RxRDY_n obsługi przerwania od sygnału RxRDY USART-a nr n,

- podprogram read_from_input_buffer_n_ (typu PUBLIC) odczytujący pojedyńczy znak (bajt) z n-tego bufora wejściowego,

- podprogram write_to_output_buffer_n_ (typu PUBLIC) wpisujący pojedyńczy znak (bajt) do n-tego bufora wyjściowego oraz inicjujący wysyłkę informacji z tego bufora,

- podprogram read_error_indicator_n_ (typu PUBLIC) odczytująca wskaźniki błędów dla n-tego kanału transmisji szeregowej.

- podprogram reset_error_n_ (typu PUBLIC) kasujący wskaźniki błędów dla n-tego kanału transmisji szeregowej.

Oprócz tego plik KANAL_n.A86 zawiera deklaracje zmiennych związanych z obsługą od strony programowej n-tego kanału pakietu MI06.

Procedury obsługi przerwań wymienione powyżej korzystają z podprogramów, które wysyłają odpowiednie sekwencje sterujące do sterowników przerwań PIC 8259A w momencie zakończenia obsługi przerwania. Podprogramy te zawarto w osobnym pliku o nazwie END_INT.A86.

2.5. System przerwań

Pakiet komunikacyjny, zrealizowany na bazie modułu MI06, obsługuje (w zależności od potrzeb) do 10 przerwań, które są generowane przez sygnały TxRDY i RxRDY każdego z pięciu układów USART 8251A. W obecnej wersji obsługa przerwań jest jednopoziomowa, tzn. rozpoczęcie obsługi danego przerwania blokuje obsługę innych, nawet tych o aktualnie wyższym priorytecie. Przerwania są ponadto blokowane programowo (instrukcja CLI) podczas wykonywania procedur "write_to_output_buffer_n_" i "read_from_input_buffer_n_" ze względu na możliwość wzajemnego nałożenia się na siebie efektów działania dwóch podprogramów modyfikujących te same liczniki obsługi danego bufora. Poszczególne

sterowniki przerwań 8259A zostały zaprogramowane do pracy w trybie z cyklicznymi priorytetami (ang. rotating priority mode), w którym wszystkie zgłoszenia są równouprawnione (aktualnie obsługiwane zgłoszenie, po zakończeniu obsługi otrzymuje najniższy priorytet).

W procedurach źródłowych przyjęto następujące numery poszczególnych przerwań:

Zrodło przerwania:	Nazwa sygnału:	Numer pola wektora przerwań:	Nazwa procedury obsługi przerwania:	Kontroler przerwań:
INT z układu PIC_0_SLAVE	INT	16	TxRDY_n n = 1..4 RxRDY_n n = 1..4	SYSTEM PIC (MASTER)
INT z układu PIC_1_SLAVE	INT	17	TxRDY_5 RxRDY_5	
-	-	18 -23	nieobsługiwane	
USART 1	TxRdy RxRdy	24 25	TxRDY_1 RxRDY_1	PIC 0 (SLAVE)
USART 2	TxRdy RxRdy	26 27	TxRDY_2 RxRDY_2	
USART 3	TxRdy RxRdy	28 29	TxRDY_3 RxRDY_3	
USART 4	TxRdy RxRdy	30 31	TxRDY_4 RxRDY_4	
USART 5	TxRdy RxRdy	32 33	TxRDY_5 RxRDY_5	PIC 1 (SLAVE)
USART 6	TxRdy RxRdy Syndet	34 35 36	SLAVE_1_IRx SLAVE_1_IRx SLAVE_1_IRx	
CLK 3	OUT 2	37	SLAVE_1_IRx	
-	-	38 - 39	SLAVE_1_IRx	

W momencie inicjacjowania poszczególnych układów modułu MI06 podprogram mi06_ini_ wypełnia:

- pola 24 - 31 wektora przerwań adresami skoku do procedury SLAVE_0_IRx (podprogram typu "NIC NIE ROB"),
- pola 32 - 39 wektora przerwań adresami skoku do procedury SLAVE_1_IRx (podprogram typu "NIC NIE ROB").

Adres właściwej procedury obsługi danego przerwania zostaje wpisany do danego pola wektora przerwań, w zależności od potrzeb, przez podprogramy "kanal_n", gdzie n $\in$ <1, 5>.

ZBIÓR PLIKÓW STANOWIĄCYCH PAKIET OBSŁUGUJĄCY KOMUNIKACJĘ OD STRONY ROBOTA

Spis treści

1. Plik find1sen.c86

Plik znajduje się w katalogu: AU\MASTER\GLOBSPAC\find1sen.c86

Komentarz:
Znalezienie adresu czujnika jednobitowego.

Wywołanie:

```
findonebit_sensor(onebit_sensor_number);
```

Dane wejściowe:

onebit_sensor_number - numer czujnika jednobitowego

Dane wyjściowe:

- Funkcja zwraca adres czujnika jednobitowego o numerze onebit_sensor_number. Jeśli nie ma czujnika jednobitowego o takim numerze to funkcja zwraca wartość NULL.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, types.h, globspac.h

2. Plik find2sen.c86

Plik znajduje się w katalogu: AU\MASTER\GLOBSPAC\find2sen.c86

Komentarz:
Znalezienie adresu czujnika dwubitowego.

Wywołanie:

```
findtwobit_sensor(twobit_sensor_number);
```

Dane wejściowe:

twobit_sensor_number - numer czujnika dwubitowego

Dane wyjściowe:

- Funkcja zwraca adres czujnika dwubitowego o numerze twobit_sensor_number. Jeśli nie ma czujnika dwubitowego o takim numerze to funkcja zwraca wartość NULL.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, types.h, globspac.h

3. Plik findasen.c86

Plik znajduje się w katalogu: AU\MASTER\GLOBSPAC\findasen.c86

Komentarz:
Znalezienie adresu czujnika analogowego.

Wywołanie:

```
findanalog_sensor(analog_sensor_number);
```

Dane wejściowe:

analog_sensor_number - numer czujnika analogowego

Dane wyjściowe:

- Funkcja zwraca adres czujnika analogowego o numerze analog_sensor_number. Jeśli nie ma czujnika analogowego o takim numerze to funkcja zwraca wartość NULL.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, types.h, globspac.h

4. Plik findvsen.c86

Plik znajduje się w katalogu: AU\MASTER\GLOBSPAC\findvsen.c86

Komentarz:
Znalezienie adresu czujnika wizyjnego.

Wywołanie:

```
findvision_sensor(vision_sensor_number);
```

Dane wejściowe:

vision_sensor_number - numer czujnika wizyjnego

Dane wyjściowe:

- Funkcja zwraca adres czujnika wizyjnego o numerze vision_sensor_number. Jeśli nie ma czujnika wizyjnego o takim numerze to funkcja zwraca wartość NULL.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, types.h, globspac.h

5. Plik findvect.c86

Plik znajduje się w katalogu: AU\MASTER\GLOBSPAC\findvect.c86

Komentarz:
Znalezienie adresu wektora korekcji.

Wywołanie:

findcorrvector(corrvector_number)

Dane wejściowe:

corrvector_number - numer wektora korekcji

Dane wyjściowe:

- Funkcja zwraca adres wektora korekcji o numerze corrvector_number. Jesli nie ma wektora korekcji o takim numerze to funkcja zwraca wartosc NULL.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, types.h, globspac.h

6. Plik getsens.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\getsens.c86

Komentarz:
Funkcja odczytująca z czujnika wartość oraz przekształcająca ją na przeskalowany współczynnik prędkości.

Wywołanie:

b = get_sens(sensor_no,typesensor,module,value);

Dane wejściowe:

char	sensor_no	- numer czujnika, z ktorego nastepuje odczyt
char	typesensor	- typ czujnika
unsigned	module	- czwarty parametr wektora korekcji (współczynnik skalujący)

Dane wyjściowe:

char	b	- wartość zwracana przez funkcję OK - gdy odczyt pomyślny nr bledu - gdy nie
double	*value	- adres zmiennej zawierającej wartość odczytu

Lista plików dołączanych (#include):

\AU\MASTER\master.h, \AU\MASTER\types.h, \AU\MASTER\errors.h

Wykorzystuje:
read_sens()

7. Plik getsenc.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\getsenc.c86

Komentarz:
Funkcja odczytująca z czujnika wartość oraz przekształcająca ją na tablicę przyrostów korekcyjnych.

Wywołanie:

```
b = get_sens(sensor_no,typesensor,vector_addr,value);
```

Dane wejściowe:

```
char      sensor_no       - numer czujnika, z którego
                            następuje odczyt
char      typesensor      - typ czujnika
COR_VECT *vector_addr[]   - adresy wektorów korekcji
                            używanych przy korekcji
                            dodatniej i ujemnej
```

Dane wyjściowe:

```
char   b          - wartość zwracana przez funkcję
                    OK        - gdy odczyt pomyślny
                    nr bledu  - gdy nie
double value[] - tablica przyrostów korekcyjnych
```

Lista plików dołączanych (#include):

\AU\MASTER\master.h, \AU\MASTER\types.h, \AU\MASTER\errors.h, \AU\MASTER\GLOBSPAC\globspac.h.

Wykorzystuje:
read_sens(), read_vis(), findvision_sensor().

8. Plik poss.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\INSTR\poss.c86

Komentarz:
"Free positioning". Wykonanie instrukcji poszukiwania swobodnego. Poszukiwanie jest realizowane tak długo, dopóki przynajmniej jeden z trzech czujników określających (przy pomocy przypisanych wektorów korekcji) przyrost położenia jest w stanie pobudzenia.

Wywołanie:

```
poss(instruction);
```

Dane wejściowe:

```
POSS_INSTR *instruction - adres instrukcji
                          programu użytkownika
```

Dane wyjściowe:

- Funkcja zwraca numer błędu lub Ø, gdy wykonanie instrukcji poszukiwania swobodnego zakończyło się sukcesem.

Zmienne zewnętrzne (extern):

tool_defined	- czy zdefiniowane narzędzie
atcpØ[]	
aorntØ[]	- aktualna pozycja i orientacja robota
aconfig	- aktualna konfiguracja ramienia robota
actual_base_velocity	- aktualna prędkość podstawowa
adaptacja	- czy włączone sa inne funkcje adaptacyjne
calcper;	- okres makrointerpolacji

lista plików dołączanych (#include):

\AU\MASTER\master.h, \AU\MASTER\INTRPRTR\types.h, \AU\MASTER\INTRPRTR\instruct.h, \AU\MASTER\errors.h, \AU\MASTER\INTRPRTR\inscodes.h, \AU\MASTER\GLOBSPAC\globspac.h, \AU\MASTER\EXTMOV\move.h

Wykorzystuje:

deactivate_sensor(), get_sens(), findonebit_sensor(), findtwobit_sensor(), findanalog_sensor(), findvision_sensor(), findcorrvector(), actlzext(), wait_for_inpos()

9. Plik contou.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\INSTR\contou.c86

Komentarz:
Włączenie konturowania.

Wywołanie:

```
contou(instruction)
```

Dane wejściowe:

CONTOU_INSTR *instruction - adres instrukcji programu użytkownika

Dane wyjściowe:

- Funkcja zwraca numer błędu lub Ø, gdy wykonanie instrukcji włączenia konturowania zakończyło się sukcesem.

Zmienne zewnętrzne (extern):

adaptacja - czy włączone są inne funkcje adaptacyjne,
contour_on - czy włączone jest konturowanie,
typesensor - typ czujnika używanego do konturowania,
vector_addr[] - tablica wskaznikow do używanych wektorow korekcji,
sensor_no - numer czujnika używanego przy konturowaniu.

Lista plików dołączanych (#include):

\AU\MASTER\master.h, \AU\MASTER\INTRPRTR\types.h, \AU\MASTER\INTRPRTR\instruct.h, \AU\MASTER\errors.h, \AU\MASTER\GLOBSPAC\globspac.h.

Wykorzystuje:

findtwobit_sensor(), findanalog_sensor(), indvision_sensor(), findcorrvector(), act_vis(), activate_sens().

10. Plik act_vis.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\act_vis.c86

Komentarz:
Aktywacja czujnika wizyjnego. Funkcja wysyła do bufora pakietu MI06 przesyłkę inicjalizacji czujnika, jeśli jest zainicjalizowane mniej niz MAX_SENS_AMOUNT czujników.

Wywołanie:

```
result = activate_vision_sens (sensor_ptr);
```

Dane wejściowe:

sensor_ptr - parametr typu *SENSOR_DESCR, będący wskaźnikiem do pola zawierającego parametry czujnika wizyjnego. Postać pola z parametrami czujnika jest identyczna jak pole definicji czujnika w 'glob_space'.

Dane wyjściowe:

- Funkcja zwraca numer błędu lub 0, gdy wykonanie aktywacji czujnika wizyjnego zakończyło się sukcesem.
result - zmienna typu char
Funkcja zwraca nastepujące wartości:
Odpowiedzi obowiązujace dla wszystkich pakietów:

SENSOR_INITIALIZED	0	- OK, czujnik został zainicjowany
BAD_SENSOR_TYPE	1	- procedura nie obsługuje tego typu czujników

Zmienne zewnętrzne (extern):

sens_count - liczba dołączonych czujników
sens_tab[] - tablica specyfikacji aktywnych czujników

Lista plików dołączanych (#include):

\AU\MASTER\INTRPRTR\sensors.h,
\AU\MASTER\INTRPRTR\sen_serv.h,
\AU\MASTER\INTRPRTR\act_sen.h,
\AU\MASTER\GLOBSPAC\globspac.h.

Wykorzystuje:

write_to_output_buffer_1(),
write_to_output_buffer_2(), write_to_output_buffer_3(),
write_to_output_buffer_4().

11. Plik read_vis.c86

Plik znajduje się w katalogu:
\AU\MASTER\INTRPRTR\read_vis.c86

Komentarz:
Funkcja odczytująca z czujnika wizyjnego wartość lub pamiętająca poprzednią wartość, jeśli nie napłynęły nowe dane z kamery.

Wywołanie:

```
b = read_vis(vision_sensor_address, value);
```

Dane wejściowe:

vision_sensor_address - wskaźnik na opis czujnika wizyjnego

Dane wyjściowe:

- Funkcja zwraca numer błędu lub 0, gdy odczytanie czujnika wizyjnego zakończyło się sukcesem.
value - adres zmiennej zawierającej wartość odczytu

Zmienne zewnętrzne (extern):

vision_counter[] - tablica liczników odczytanych znaków komunikatu

Lista plików dołączanych (#include):

\AU\MASTER\master.h, \AU\MASTER\types.h,
\AU\MASTER\errors.h.

Wykorzystuje:

read_from_input_buffer_1(),
read_from_input_buffer_2(), read_from_input_buffer_3(),
read_from_input_buffer_4(), long b3tol().

12. Plik deact mi.c86

Plik znajduje się w katalogu: \AU\MASTER\INTRPRTR\deact_mi.c86

Komentarz:
Dezaktywacja wszystkich czujników związanych z pakietami MI-Ø6. Do buforów wszystkich kanałów pakietu MI-Ø6 wysyłana jest przesyłka o informująca o dezaktywacji.

Wywołanie:

```
deact_mi();
```

Dane wejściowe: - brak

Dane wyjściowe: - brak

Zmienne zewnętrzne (extern): - brak

Wykorzystuje:

write_to_output_buffer_1(), write_to_output_buffer_2(), write_to_output_buffer_3(), write_to_output_buffer_4().

13. Plik linear.c86

Plik znajduje się w katalogu: AU\MASTER\EXTMOV\linear.c86

Komentarz:
Wykonanie ruchu po linii prostej z jednostajną mianą orientacji osi narzedzia. W przypadku, gdy Włączone jest konturowanie ruch narzędzia jest modyfikowany tak, aby uwzględnić informacje uzyskane z czujnika.

Wywołanie:

```
linear()
```

Dane wyjściowe:

wartość funkcji:

LINEAR_SLOW	- za wolny ruch
LINEAR_CONSTRAINTS	- przekroczonie ograniczeń
LINEAR_VELOCITY	- przekroczonie ograniczeń prędkości
LINEAR_INSTR_INT	- instrukcja została przerwana
LINEAR_CONFIG	- zmiana konfiguracji
LINEAR_ERROR	- inne błędy

Zmienne zewnętrzne (extern):

VECTOR rtcpØ - Docelowe położenie punktu

TCP [mm]
VECTOR rorntØ - Docelowa orientacja osi narzędzia (wektor jednostkowy)
int rconfig - Zadana konfiguracja ramienia robota
double rpt - Czas ruchu [s]
VECTOR atcpØ - Aktualne położenie punktu TCP [mm]
VECTOR aorntØ - Aktualna orientacja osi narzędzia (wektor jednostkowy)
VECTOR dtcpØ - Wektór przyrostu położenia punktu TCP (= rtcpØ - atcpØ)
double calcper - Okres obliczeniowy [s]
BOOLEAN oscillation - określa czy występują oscylacje
VECTOR oscillation_vector - Wektór oscylacji
double oscillation_deflection - Aktualna wartość wychylenia
int oscillation_nosteps - Liczba kroków potrzebna dla wykonania połowy jednej oscylacji
VECTOR tcpØ, orntØ - pośrednia poz. robota we współrzędnych zewnętrznych
double lambda - = step / steps
VECTOR stcpØ, sorntØ - początkowa pozycja robota w ruchu prostoliniowym we współrzędnych zewnętrznych
int aconfig - aktualna konfiguracja ramienia robota
SENSOR sensor[] - tablica czujników
BOOLEAN contour_on - konturowanie włączone (1) wyłączone (Ø)
int sensor_no - numer czujnika używanego do konturowania
char typesensor - typ czujnika używanego do konturowania
COR_VECTOR *vector_addr[] - adresy wektorów korekcji używanych do konturowania

Lista plików dołączanych (#include):

AU\MASTER\master.h, AU\MASTER\intrprtrtypes.h, linear.h, move.h

Wykorzystuje:

vectcopy(), frndint1(), orntinit(), intrrptd(), vectnrml(), interm(), calculate_oscillation_deflection(), move(), char get_senc(), void deactivate_sens(), void deact_mi(),

Indeks

4. Badania współpracy układu sterowania robota IRp z czujnikiem wizyjnym 1D.

4.1. Wstęp.

Czujnik wizyjny 1D został opracowany przez OAE w ramach zlecenia RP-63. Został on przekazany do OAP w dniu 1990.01.31. Podczas przekazywania czujnika, OAE zademonstrował jego sprawność z oprogramowaniem testującym - w kasecie INTELDIGIT PROWAY. Badania współpracy czujnika wizyjnego 1D z robotem zostały zakończone w dniu 1990.02.28, jednak ze względu na powstałą awarię czujnika oraz konieczność wykonywania innych prac planowanych w OAP, przesunięto termin odbioru etapu 1 zlecenia 9532 na dzień 1990.04.20.

4.2. Przebieg badań.

W trakcie badań funkcjonalnych czujnika 1D, w czujniku tym dokonywanych było szereg zmian programowych, których konieczność wprowadzenia wynikła dopiero po dołączeniu czujnika do robota. Na początku badań usunięto zjawisko odsyłania przesyłek skierowanych z robota do czujnika.

Dalsze badania wykazały, że znaki wysyłane do czujnika są przez niego gubione. W celu możliwości takiego stwierdzenia OAP opracował specjalny program testujący, który umożliwił określenie przyczyny tego zjawiska. Próby wykazały, że oprogramowanie obsługujące część odbiorczą czujnika nie nadąża za sygnałami przychodzącymi z robota.

Z przeprowadzonej dyskusji z autorem oprogramowania wynika, że w celu usunięcia tej niedogodności należałoby przeprogramować bufor czujnika z uwzględnieniem wymaganych szybkości transmisji. W stosunku do dokumentacji oprogramowania czujnika (Nr 6291 - zał. Nr 4) wprowadzono zmiany znaczenia komunikatów.

Niżej wymienionym funkcjom przypisano następujące znaczenia:

POSA_ACT - pytanie o dane z pierwszej kamery,
POSB_ACT - pytanie o dane z drugiej kamery,

POSC_ACT - pytanie o dane z trzeciej kamery.

Komunikatom

SET_FLAG i CUR_POS nie zostały przypisane żadne znaczenia. Format komunikatu S_ERROR pozostał bez zmian. Pierwotnie przyjęte przesyłanie liczb zmiennoprzecinkowych okazało się niekorzystne i zostało zamienione na przesyłanie liczb całkowitych z zakresu 0÷255. Dzięki temu komunikaty: POZA_REC, POZB_REC, POZC_REC (dane z pierwszej, drugiej lub trzeciej kamery) skrócono do niezbędnej długości. Po wprowadzonych zmianach stwierdzono bezbłędny odczyt treści komunikatów.

W czasie dalszych badań stwierdzono, że odczyt danych przekazywanych z kamery jest niewłaściwy. Dane te nie odpowiadają położeniom rzeczywistym kamery i nie zmieniają się przy zmianie usytuowania kamery w stosunku do otoczenia.

4.3. Wnioski z badań.

W czasie badań funkcjonylnych czujnika wizyjnego stwierdzono, że:

- jest komunikacja pomiędzy sterownikiem, a czujnikiem. Treści komunikatów są odczytywane prawidłowo,
- szybkość obsługi informacji odbieranych przez czujnik jest niewystarczająca,
- dane przekazywane przez czujnik nie są skorelowane z jego położeniem w stosunku do otoczenia.

5. Ocena wykorzystania wyników pracy własnej realizowanej wg zlecenia 9532.

Ten punkt sprawozdania wprowadzono po zakończeniu tematu 1122. W wyniku odbioru pracy 1122, przy zastosowaniu czujnika wizyjnego opracowanego przez Instytut Automatyki Przemysłowej Politechniki Warszawskiej, komisja przyjęła i potwierdziła prawidłowość wykonania tematu pt.: "Realizacja oprogramowania obsługi inteligentnych układów sensorycznych". Oceny dokonano podczas praktycznego zademonstrowania działającego oprogramo-

wania w pracującym układzie robota. Tym samym potwierdzono prawidłowość wykonania tematu pracy własnej 9532, której zakres obejmował zapewnienie komunikacji pomiędzy czujnikiem wizyjnym, a układem sterowania robota.

Odbioru końcowego pracy 1122 dokonano w dniu 91.01.24.

91.01.24

91.01.25