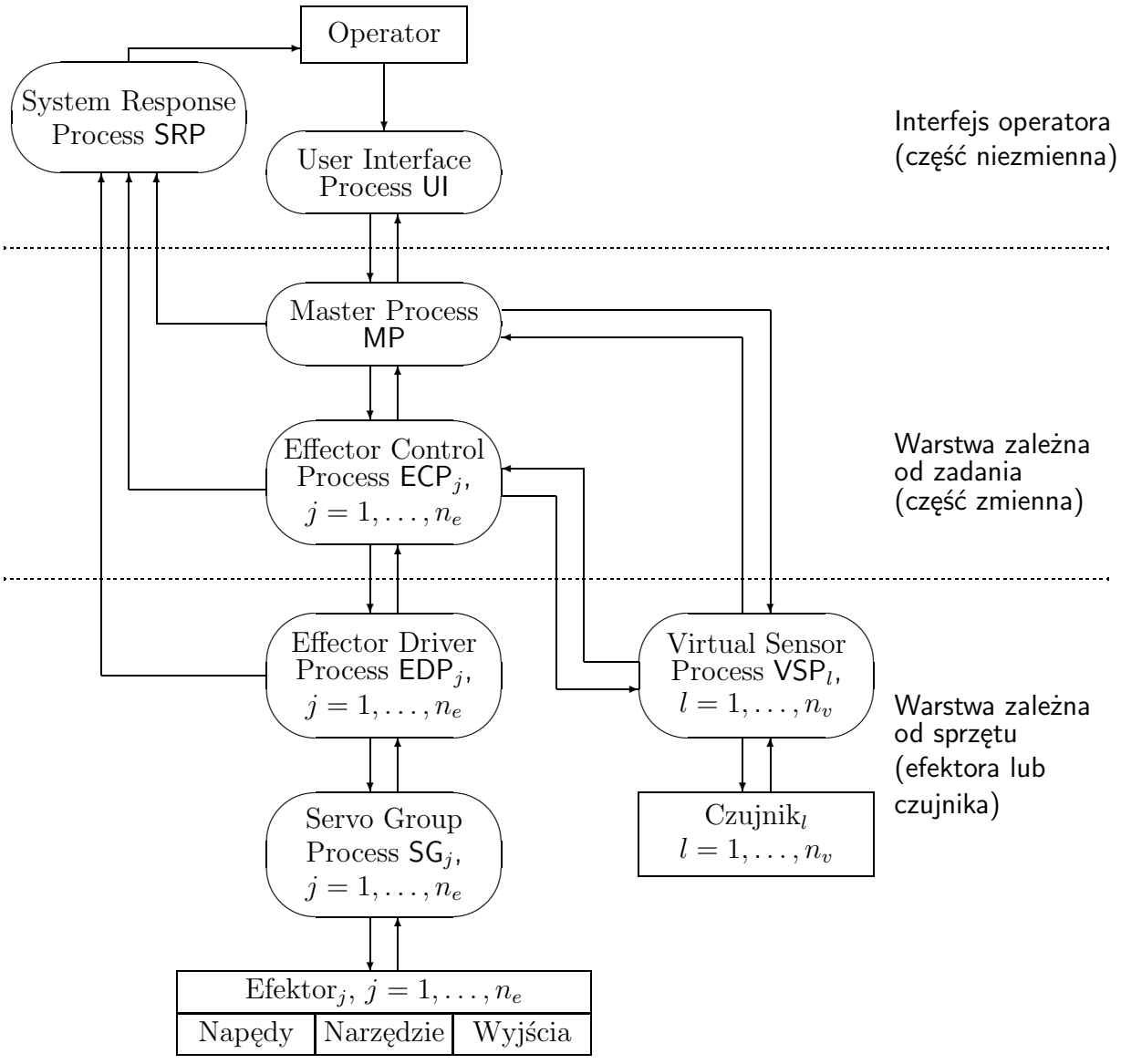
Wojciech Szynkiewicz
Instytut Automatyki i Informatyki Stosowanej PW
Zespół Robotyki

Uruchamianie i obsługa systemu MRROC++

1. Ogólna struktura systemu MRROC++

System MRROC++ jest jednocześnie biblioteką modułów służących do konstruowania sterowników programowych dedykowanych zadaniom, które ma wykonać robot lub system wielorobotowy, oraz językiem programowania, w którym to zadanie jest opisywane (programowane.) Konkretne sterowniki MRROC++ mają hierarchiczną strukturę funkcjonalną i składają się z trzech warstw: zależnej od sprzętu, zależnej od zadania oraz interfejsu operatora (Rys. 1).



Rys. 1: Struktura systemu MRROC++

Poszczególne funkcje sterownika realizowane są przez moduły w postaci odrębnych procesów działających w węzłach sieci lokalnej (możliwe jest działanie na jednym lub kilku komputerach) pod kontrolą systemu operacyjnego QNX 4.2x.
W skład interfejsu operatora wchodzą dwa procesy User Interface Process (UI) oraz System Response Process (SRP).

- Proces UI umożliwia obsługę kompletnego sterownika (np. ładowanie/zabijanie procesów, uruchamianie/zatrzymywanie, synchronizacji robota, wybór węzłów do uruchamiania procesów, zmianę parametrów).
- Proces SRP wyświetla komunikaty: o stanie efektora (robota lub innego urządzenia) i sterownika oraz wykrytych błędach.

Proces MP jest koordynatorem działania wszystkich współpracujących efektorów (jeśli jest ich jest więcej niż jeden). Jeśli sterownik dedykowany jest tylko jednemu efektorowi (tzn. sterujemy pojedynczym robotem) wówczas MP jest przezroczysty i przesyła jedynie polecenia z UI do niższych warstw.
Sterownik każdego efektora (robota lub innego urządzenia) przeznaczony do realizacji określonego zadania składa się z trzech procesów:

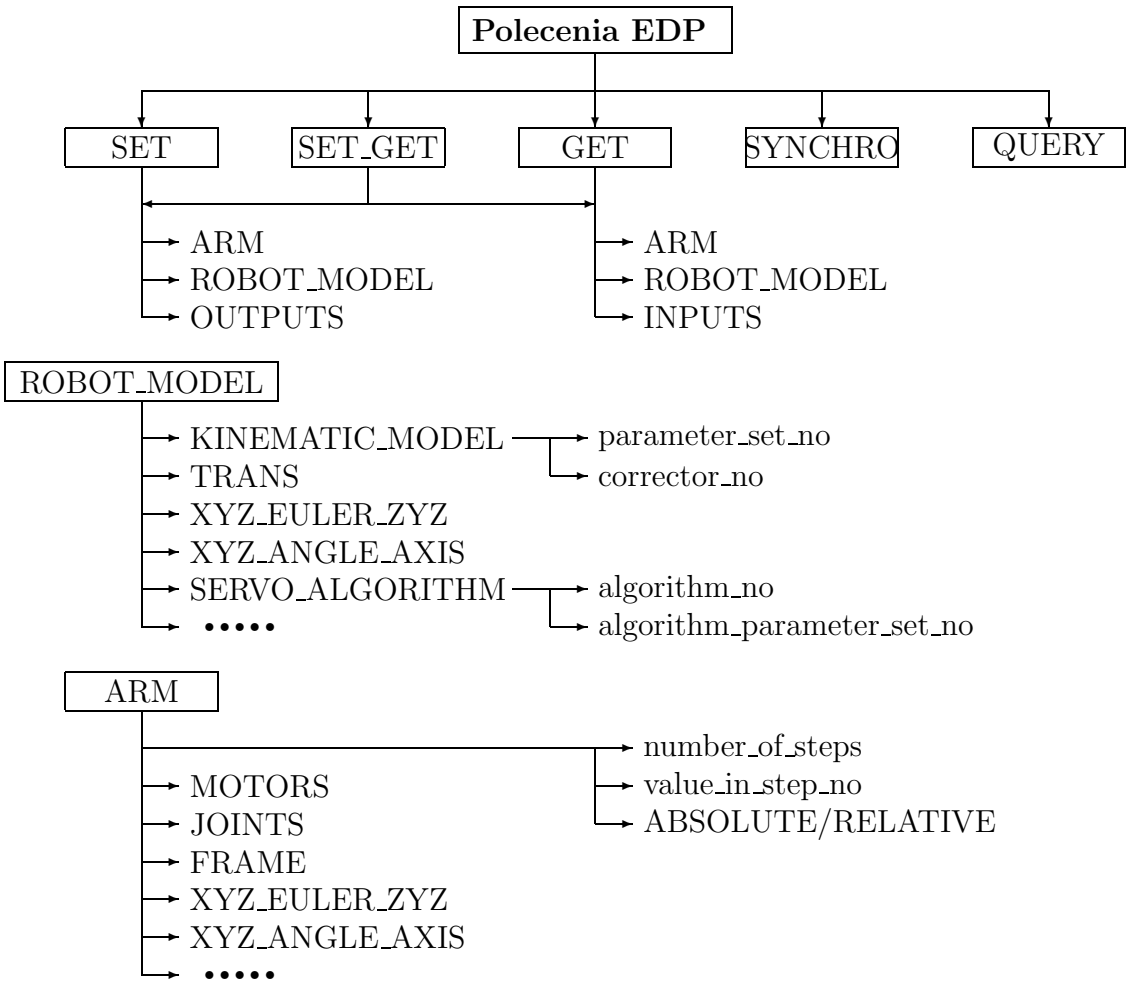
- Effector Control Process ECP,
- Effector Driver Process EDP
- Servo Group Process SG.

Napisanie nowego zadania użytkownika wymaga modyfikacji fragmentów kodów źródłowych procesów MP oraz ECP. Program użytkowy jest napisany w C++ z użyciem funkcji bibliotecznych systemu MRROC++. Realizuje zadanie zlecone systemowi do wykonania przez użytkownika. Program użytkowy stanowi jądra procesów: MP i ECP. Najczęściej program użytkowy skonstruowany jest jako nieskończona pętla (`for(;;)`), którą objęte są instrukcje (funkcje) `Move` i `Wait` oraz dodatkowe instrukcje języka C++. Cała zmienność instrukcji, wynikająca z różnorodności wykonywanych ruchów, została skoncentrowana w dwóch parametrach: generatorze trajektorii (dla `Move`) oraz warunku początkowym (dla `Wait`). Stałą liczbę parametrów instrukcji ruchowych uzyskano przez zastosowanie list robotów i czujników.
EDP jest właściwym sterownikiem konkretnego typu robota (lub innego urządzenia). Jest to serwer wykonujący polecenia klientów (procesów ECP oraz UI). Zmiana typu robota wymaga wymiany EDP. Zestaw poleceń dla EDP jest pokazany na Rys.2.
Polecenia są szczegółowo opisane w podręczniku (1).
Dane odczytywane z czujników rzeczywistych są przetwarzane w Virtual Sensor Process (VSP), które można traktować jako sterowniki czujników.
Obsługa sytuacji awaryjnych została oddzielona od kodu sterującego normalnym wykonaniem instrukcji i jest realizowana jako obsługa wyjątków.
Kontakt z warstwą sprzętową sterownika realizowany jest przez proces Servo Group Process (SG), realizuje on także serwomechanizmy poszczególnych osi robota.

2. Obsługa sprzętu

2.1. Sterowniki osi

Długość cyklu regulacji (kroku sterowania) może być ustawiona na wartość $n \times 0.512$ms, gdzie $n = 1, 2, \ldots$. W przypadku robota IRp-6 na torze jezdnym przyjęto krok równy 4×0.512ms (ustawienie domyślne). Moment odebrania *proxy* (wyzwalanego cyklicznie przez funkcję obsługi przerwań – przerwanie IRQ10) określa rozpoczęcie kolejnego kroku. W



Rys. 2: Polecenia wykonywane przez EDP

każdym kroku regulacji następuje kontakt z warstwą sprzętową sterownika. Bezpośrednie odwołania do rejestrów sprzętowych znajdują się w jednej funkcji `read_write_hardware()`. W skład warstwy sprzętowej wchodzi sześć mikroprocesorowych sterowników osi. Sterownik każdej osi widziany jest przez komputer nadrzędny (PC) jako grupa 16-bitowych rejestrów wejścia-wyjścia dostępnych pod adresami $0x21n$ (wspólnymi dla wszystkich osi). Numer sterownika osi, którego będą dotyczyły przesłania, musi być przedtem zapisany do 8-bitowego rejestru o adresie $0x315$. Format numeru sterownika jest następujący (tabela 1):

$$11bbbbbb$$

gdzie *bbbbbb* – numer binarny sterownika:

oś	numer binarny	Człon
0	000000	tor jezdny
1	000001	kolumna obrotowa
2	000010	ramię dolne
3	000011	ramię górne
4	000100	pochylenie kiści
5	000101	obrót kiści

Tabela 1: Numery binarne sterowników osi i człony robota IRp-6 na torze

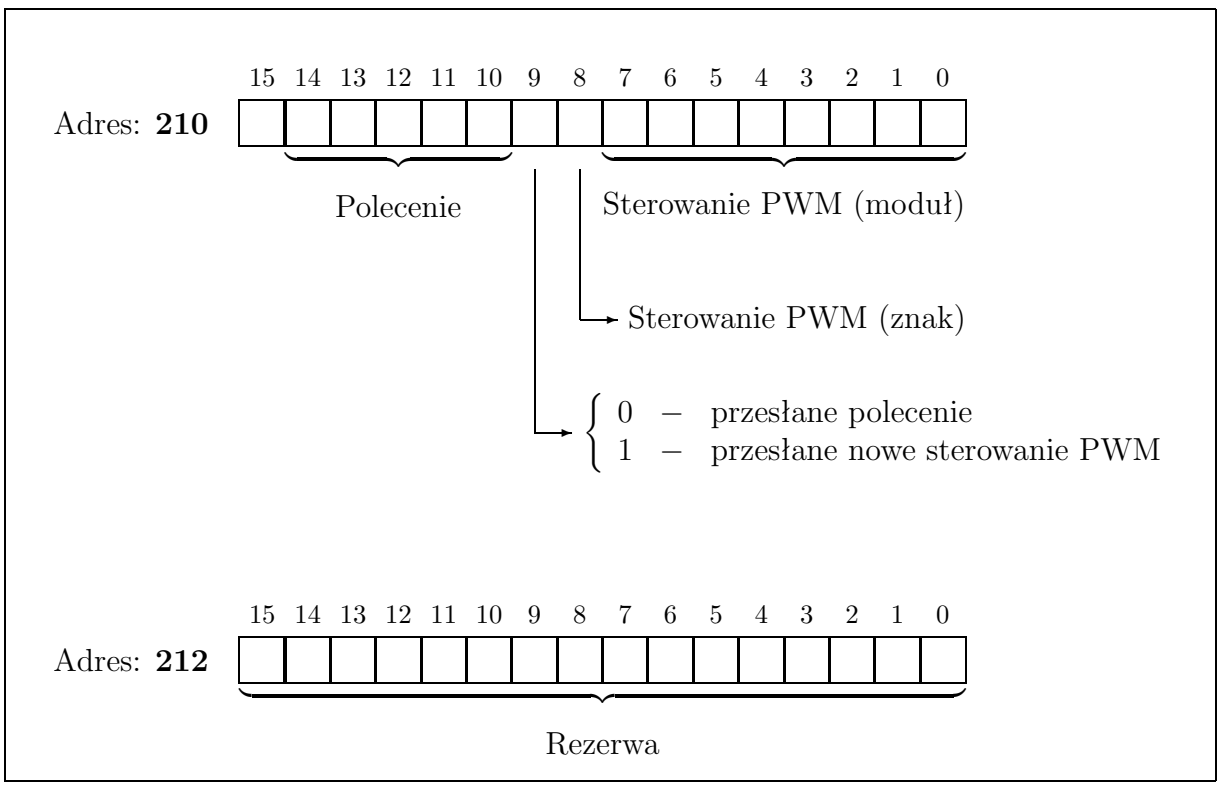
Uwaga: Odwołanie do adresu $0x21n$, przy wpisanym do rejestru o adresie $0x315$ numerze nieistniejącej osi spowoduje zawieszenie komputera.
Sterowniki mają zatem przypisane kolejne numery (w postaci szesnastkowej) $0xC1$, $0xC2$,. . ., $0xC6$. Każdy sterownik osi ma następujące rejestry wejścia-wyjścia (tabela 2):

Rejestr	Adres
SERVO_COMMAND_1_ADR	$0x210$
SERVO_COMMAND_2_ADR	$0x212$
SERVO_REPLY_STATUS_ADR	$0x210$
SERVO_REPLY_INT_ADR	$0x212$
SERVO_REPLY_POS_LOW_ADR	$0x214$
SERVO_REPLY_POS_HIGH_ADR	$0x216$
SERVO_REPLY_REG_1_ADR	$0x218$
SERVO_REPLY_REG_2_ADR	$0x220$

Tabela 2: Nazwy i adresy rejestrów wejścia/wyjścia sterowników osi

2.1.1. Rejestry wyjściowe – do zapisu przez komputer nadrzędny

Format danych przesyłanych z komputera nadrzędnego do sterowników osi przedstawia rys.3.



Rys. 3: Format danych przesyłanych z komputera do sterowników osi

Zapisywane do rejestru SERVO_COMMAND_1_ADR pod adres $0x210$ mają następujące znaczenie:

- bity 0..7 – wartość bezwzględna sterowania będącego wypełnieniem fali prostokątnej (PWM) z zakresu 0–255,

- bit 8 – znak sterowania:
  $\begin{cases} 0 & - \text{ dodatnie} \\ 1 & - \text{ ujemne} \end{cases}$
- bit 9 – wybór sterowanie/polecenie:
  $\begin{cases} 0 & - \text{ przesłano nowe polecenie} \\ 1 & - \text{ przesłano nowe sterowanie} \end{cases}$
- bity 10..14 – rodzaj polecenia (tabela 3):

Polecenie	Kod	Opis
RESET_POSITION_COUNTER	$0x0400$	Zerowanie liczników położenia
RESET_MANUAL_MODE	$0x0800$	Zerowanie pracy ręcznej
RESET_ALARM	$0x0C00$	Zerowanie alarmu sytuacji awaryjnej
PROHIBIT_MANUAL_MODE	$0x1000$	Zakaz pracy ręcznej
ALLOW_MANUAL_MODE	$0x1400$	Zezwolenie na pracę ręczną
START_SYNCHRO	$0x1800$	Rozpoczęcie synchronizacji
FINISH_SYNCHRO	$0x1C00$	Zakończenie synchronizacji osi
SET_INT_FREQUENCY	$0x20dd$	Ustaw dzielnik częstotliwości przerwań
SET_MAX_CURRENT	$0x24pp$	Ustaw prąd maksymalny

Tabela 3: Polecenia dla sterowników mikroprocesorowych osi (kod szesnastkowy słowa wpisanego pod adres 0x210); dd – wartość dzielnika przerwań (przerwania są generowane co $0.512 \times dd$ ms – domyślnie $dd = 4$); pp – wartość prądu maksymalnego silnika ($1A \approx 13.7$ jednostki – domyślnie $pp = 40 \Rightarrow I_{max} = 4A$)

Rejestr o adresie $0x212$ jest rezerwą przeznaczoną do wykorzystania przy rozszerzeniu możliwości sterowników osi o wbudowane regulatory PID.

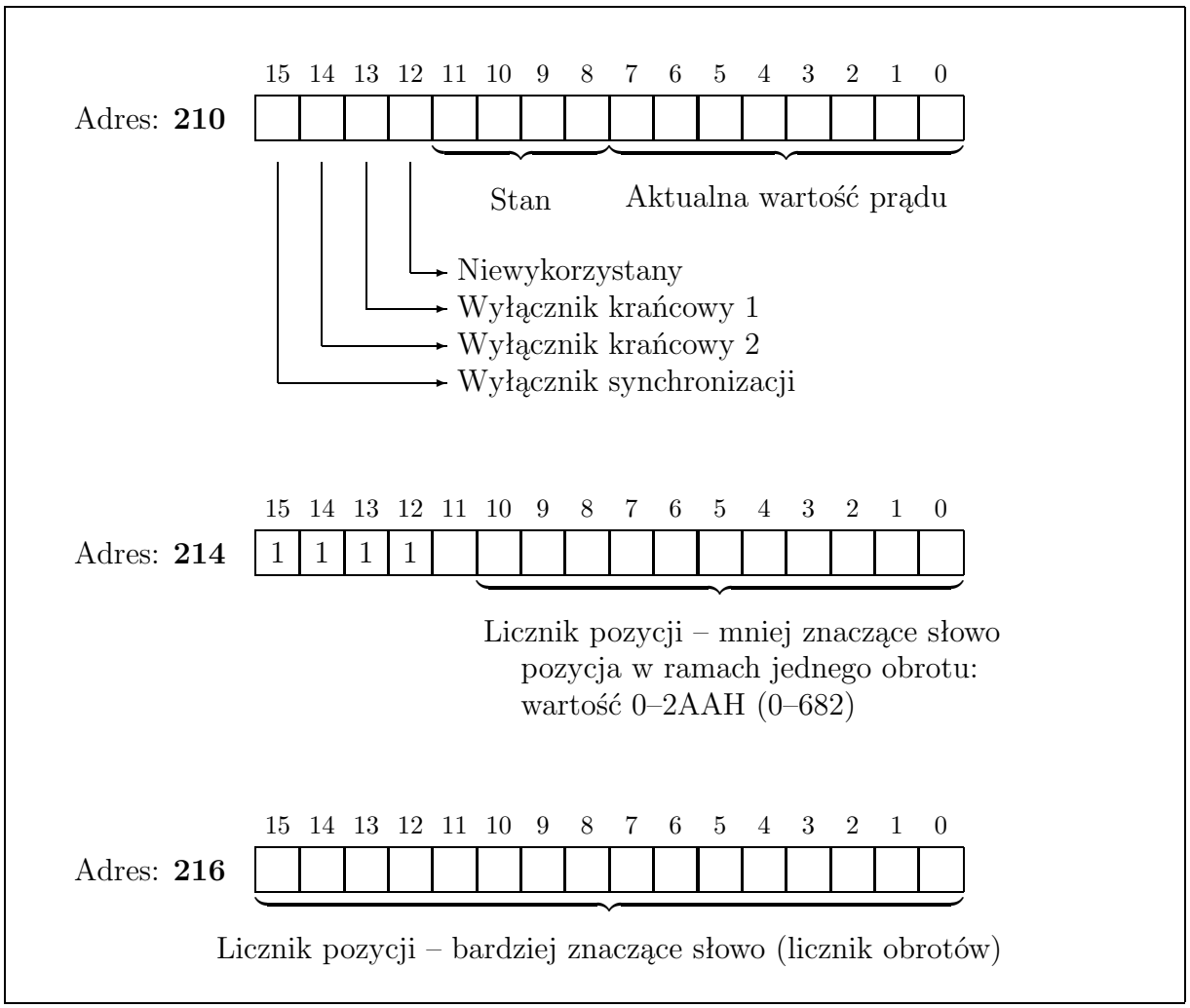
2.1.2. Rejestry wejściowe – do odczytu przez komputer nadrzędny

Na rys.3 przedstawiono format danych odczytywanych z rejestrów sterownika osi przez komputer nadrzędny.
Dane odczytane z rejestru SERVO_STATUS_ADR (adres $0x210$) mają następujące znaczenie:

- bity 0..7 – aktualna wartość prądu,
- bit 8..11 – stan sterownika,
- bit 12 – niewykorzystany
- bit 13 – stan wyłącznika krańcowego dolnego:
  $\begin{cases} 0 & - \text{ rozwarty (najechano na wyłącznik krańcowy)} \\ 1 & - \text{ normalnie zwarty,} \end{cases}$
- bit 14 – stan wyłącznika krańcowego górnego:
  $\begin{cases} 0 & - \text{ rozwarty (najechano na wyłącznik krańcowy)} \\ 1 & - \text{ normalnie zwarty,} \end{cases}$

Bit	Wartość	Znaczenie
8	1	odebrano poprawne polecenie (przy synchronizacji – zero rezolwera)
9	1	stan STOP zadziałał wyłącznik krańcowy
10	1	stan STOP przeciążenie prądowe
11	1	ustawiona praca ręczna

Tabela 4: Stan sterownika osi

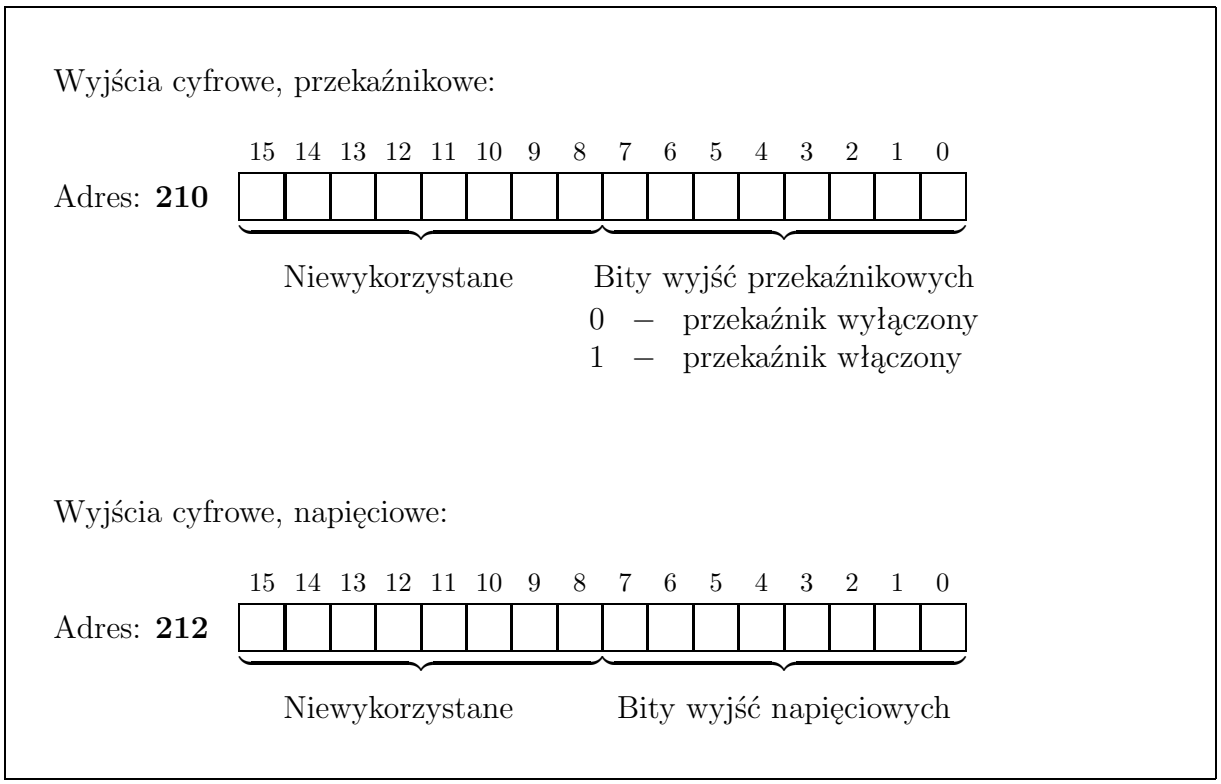


Rys. 4: Format danych przesyłanych ze sterowników osi do komputera

- bit 15 – stan wyłącznika synchronizacji,
  $\begin{cases} 1 & - \text{ zwarty (najechano na wyłącznik synchronizacji)} \\ 0 & - \text{ normalnie rozwarty,} \end{cases}$

Rejestry SERVO_REPLY_POS_LOW_ADR ($0x214$) oraz SERVO_REPLY_POS_HIGH_ADR ($0x216$) zawierają bieżące odczyty liczników położenia wału silnika: odpowiednio młodszego (mniej znaczącego) słowa i starszego (bardziej znaczącego słowa). Do obliczenia rzeczywistego położenia należy zanegować wszystkie bity obu słów. Z młodszego słowa należy wykorzystać tylko 11 najmłodszych bitów (wyciąć 5 najstarszych bitów). Położenie jest mierzone w impulsach rezolwera. Na jeden pełny obrót wału przypadają 682 impulsy. W trakcie cyklicznej realizacji (w nieskończonej pętli) są wykonywane, kolejno dla wszystkich sterowników osi, następujące czynności:

1. Sprawdzenie stanu wyłączników synchronizacji, wyłączników krańcowych i w zależności od stanu tych wyłączników ustawienie zmiennej `hardware_error`:
   (a) `hardware_error = UPPER_LIMIT_SWITCH` – gdy rozwarty jest wyłącznik krańcowy górny,
   (b) `hardware_error = LOWER_LIMIT_SWITCH` – gdy rozwarty jest wyłącznik krańcowy dolny,
   (c) `hardware_error = OVER_CURRENT` – gdy wystąpiło przeciążenie prądowe,
   (d) `hardware_error = SYNCHRO_SWITCH_ON` – gdy zwarty jest wyłącznik krańcowy dolny,



Rys. 5: Rejestry wyjściowe

2. Odczyt bieżącej wartości położenia bezwzględnego i obliczenie przyrostu położenia wału silnika.
3. Jeśli nie było błędu (`hardware_error = 0`) zapis wartości zadanej wypełnienia fali PWM i określenie nowego kierunku obrotu wału.

Wyżej wymienione czynności wykonywane są w każdym kroku sterowania (przedziale dyskretyzacji).
UWAGA: Wyłączniki krańcowe w robocie IRp-6 na torze jezdnym nie są jeszcze podłączone.

2.2. Układy wejścia-wyjścia

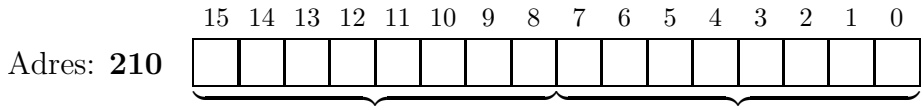
Pakiet wejścia-wyjścia jest widziany przez komputer nadrzędny jako grupa 16-bitowych rejestrów wejścia-wyjścia dostępnych pod adresami $0x21n$ (podobnie jak sterowniki osi). Numer binarny pakietu 11001000 musi być przedtem zapisany do 8-bitowego rejestru o adresie $0x315$. Adresowanie rejestrów na pakiecie wyjścia-wyjścia jest jest następujące:
• Rejestry wyjściowe (do zapisu przez PC) rys. 5.
• Rejestry wejściowe (do odczytu przez PC) rys. 6.
UWAGA: Przy przesyłaniu danych między pakietem wejścia-wyjścia a PC dane są negowane.

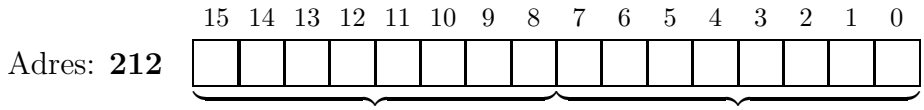
3. Struktura katalogów oraz zawartość plików

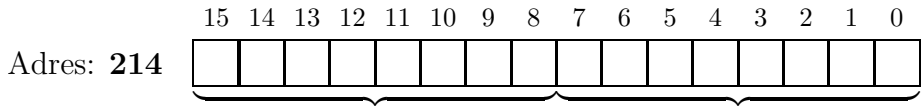
Struktura katalogów pakietu MRROC++ jest następująca: Katalog główny o nazwie `mrrocpp` może znajdować w dowolnym miejscu w struktury katalogów systemu QNX 4.2x. Struktura podkatalogów `mrrocpp` z UI działającym w QNX Windows jest pokazana na rys. 7:

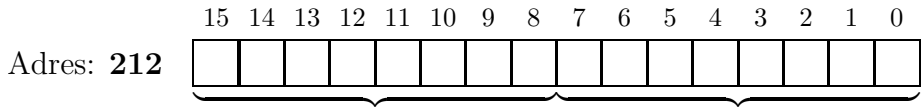
- Pliki wykonywalne oraz konfiguracyjne (np. `ui.cfg`) są w katalogu `bin`.

Wejścia analogowe (osiem kanałów, zakres 0 – +5V):

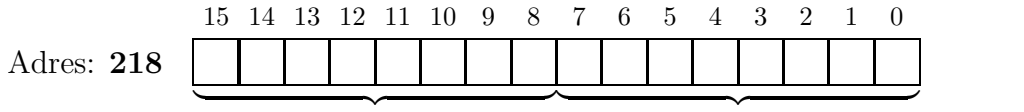




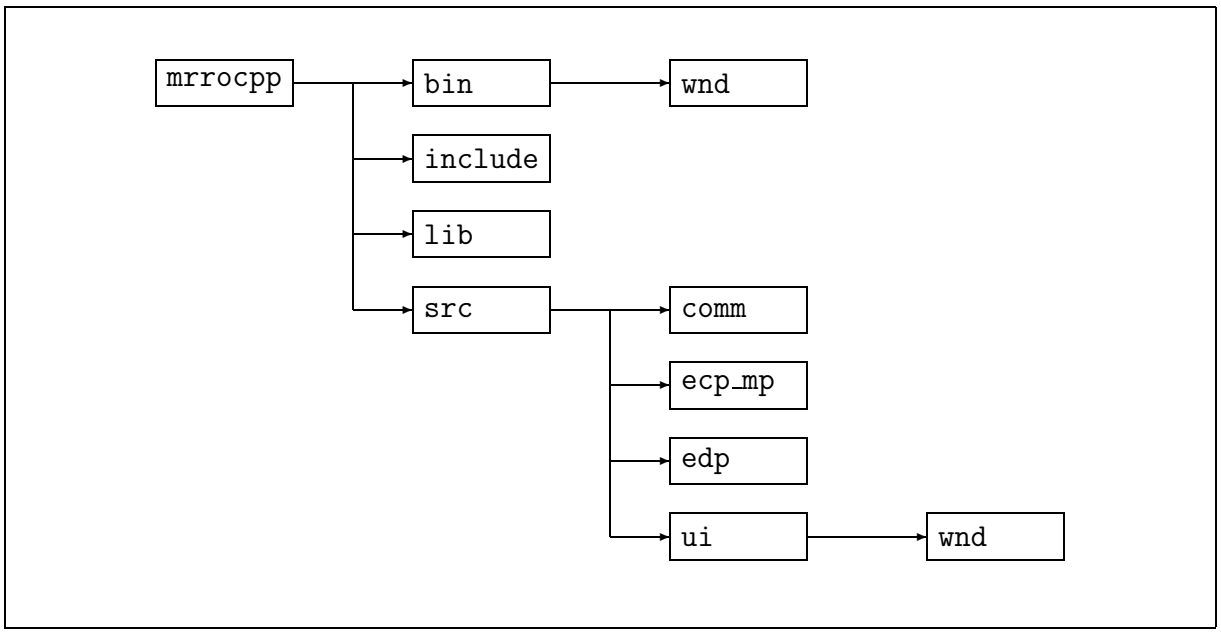




Wejścia cyfrowe:



Rys. 6: Rejestry wejściowe



Rys. 7: Struktura katalogów

- Wszystkie pliki nagłówkowe znajdują się w katalogu `include`.
- Katalog `lib` zawiera biblioteki funkcji wykorzystywanych do komunikacji międzyprocesowej systemu MRROC++.
- Pliki źródłowe poszczególnych procesów są w odpowiednich podkatalogach katalogu `src`.

4. Uruchamianie sterownika MRROC++

Najprostszy i najszybszy sposób stworzenia nowego sterownika dedykowanego konkretnemu zadaniu polega na skopiowaniu już istniejących plików źródłowych procesów ECP i MP, które realizują najbardziej podobne zadanie. Następnie należy je tak zmodyfikować, aby powstał sterownik wykonujący nowe zadanie.
W trakcie uruchamiania nowego sterownika nieuniknione są błędy. Aby je wykryć i usunąć można korzystać z wydruków kontrolnych w dowolnym miejscu programu użytkowego (z wyjątkiem procedury obsługi przerwania). Inną metodą jest przesłanie wiadomości (message) procesowi SRP, który wyświetla tę wiadomość w okienku komunikatów. Można to wykonać za pomocą funkcji `msg->message` (patrz podręcznik programowania (2)). Ponadto można przy uruchamianiu korzystać z debuggera `wd`, lecz należy pamiętać o ostrych wymaganiach czasowych, jakie muszą być spełnione przy pracy z rzeczywistym robotem.
W celu uruchomienia i pracy z istniejącym sterownikiem konkretnego typu robota (tutaj IRp-6 na torze jezdnym) i realizującego określone zadanie z UI działającym pod QNX Windows (istnieje również wersja interfejsu okienkowego działająca w środowisku graficznym Photon) należy wykonać następujące czynności:

- Należy uruchomić robota:
  1. Włączyć zasilanie modułów sterowników osi znajdujących się na stojaku (wcisnąć bezpiecznik automatyczny na dole stojaka) – powinny zaświecić się diody LED na wszystkich płytach czołowych pakietów sterowników.
  2. Włączyć zasilanie szafy sterowniczej przekręcając kluczyk na panelu na drzwiach (zgodnie z ruchem wskazówek zegara), następnie nacisnąć żółty przycisk (GO-

TOWOŚĆ) zasilania układów elektronicznych oraz przycisk (PRACA) włączający zasilanie silników napędowych osi robota.

3. Wyzerować wszystkie sterowniki osi naciskając przycisk zerowania (na górze płyty czołowej pakietu) na pakiecie interfejsu z komputerem (lewy skrajny pakiet na stojaku)
4. Po uruchomieniu robota można ręcznie poruszać w obu kierunkach poszczególnymi osiami robota za pomocą przycisków na pakietach sterowników.

- Uruchomić sterownik programowy wykonując następujące czynności:
  1. W katalogu `../mrrocpp/bin` uruchomić QNX Windows poleceniem
     `$ windows`
  2. W QNX Windows otworzyć okno terminala `wterm` (nacisnąć prawy przycisk myszki i wybrać z menu odpowiednią opcję) i uruchomić w nim proces UI poleceniem:
     `$ ./ui`
  3. Po uruchomieniu interfejsu okienkowego można korzystać z odpowiednich poleceń dostępnych dla użytkownika (menu jest kontekstowe i dostępność opcji zależy od stanu robota i sterownika) zgodnie z opisem przedstawionym w (1).
  4. Załadować i uruchomić proces EDP przez kliknięcie odpowiedniej ikony (EDP przejmuje kontrolę na robotem – nie można sterować za pomocą przycisków na pakietach sterowników). Uruchomienie EDP powoduje też uruchomienie procesu SG realizującego w sposób programowy serwomechanizmy i bezpośredni kontakt ze sprzętowymi sterownikami osi robota. Istotne jest aby proces EDP był uruchomiony w tym komputerze, w którym jest zainstalowana karta interfejsu ze sterownikami sprzętowymi osi. Po prawidłowym załadowaniu procesu EDP w okienku SRP powinien wyświetlić się komunikat:
     `EDP loaded`.
  5. Uruchomić synchronizację robota: wybierając ikonę ruchy ręczne, a następnie opcję synchronizacji robota. Pozycja startowa ramion robota musi być taka aby możliwa była synchronizacja (najazd na wyłączniki synchronizacji musi być z określonego kierunku). Po prawidłowej synchronizacji w okienku SRP powinien wyświetlić się komunikat:
     `Robot synchronised`.
  6. Po poprawnym zsynchronizowaniu robota załadować proces MP przez kliknięcie ikony i wybraniu pliku wykonywalnego `mp_m`. Spowoduje to załadowanie także procesu ECP
  7. Wykonanie zadania rozpocznie się po wysłaniu polecenia START (kliknięcie ikony)
  8. Można wstrzymać wykonanie programu opcja PAUSE i wznowić RESUME lub zatrzymać STOP
  9. Po zakończeniu zadania można usunąć proces MP. Usunięcie procesu EDP powoduje konieczność ponownej synchronizacji robota.

Wystąpienie jakichkolwiek błędów jest sygnalizowane stosownym komunikatem wyświetlanym w okienku SRP. Po prawidłowym uruchomieniu sterownika powinny działać (w jednym węźle lub w kilku) następujące procesy (z przykładowymi nazawami globalnymi):

Plik wykonywalny	Nazwa globalna
ui	
srp	/IRP6/SRP
mp_m	/IRP6/MP
ecp_m	/IRP6/ECP
edp_m	/IRP6/EDP
servo	/IRP6/SERVO_GROUP
reader	/IRP6/READER

UWAGA: Każdy robot ma przycisk alarmowy (powinien on znajdować się pod ręką), który należy wcisnąć w przypadku niepożądanego zachowania robota (powoduje to wyłączenie zasilania silników).

5. Kompilacja programów

W systemie operacyjnym QNX 4.2x do kompilacji programów napisanych w językach `C` i `C++` służą kompilatory i konsolidator Watcom 10.6. Dla ułatwienia kompilacji i uruchomienia procesów składowych pakietu `MRROC++` przygotowano zestaw plików `makefile` dla programu `make`.

5.1. Plik makefile

W każdym katalogu zawierającym pliki źródłowe lub nagłówkowe znajduje się odpowiedni plik `makefile`. Uruchomienie programu `make` z aktualnego katalogu (np. `/home/mrrocpp/-src/ecp_mp` powoduje interpretację makropoleceń umieszczonych w pliku `makefile` znajdującym się w danym katalogu. Użytkownik modyfikuje kod źródłowy programów ECP oraz MP. Zawartość przykładowego pliku `makefile` do obsługi (kompilacja, konsolidacji, instalacji, kopiowania, itp.) tych programów przedstawiono poniżej:

```
# Tworzy: EFFECTOR CONTROL PROCESS (ECP) i MASTER PROCESS (MP)
#
CFLAGS=-4 -ms
WPP=wpp386
LDFLAGS=-b -g -N 32000
WPPFLAGS=-4 -xs -Otax -ms
WPPFLAGS += -i=/home/mrrocpp/include
INCLUDE=/usr/include
LIB=/usr/lib
MRROCPPPATH = /home/mrrocpp
HEADERS = ../../include/typedefs.h  ../../include/impconst.h ../../include/com_buf.h
          ../../include/lst.h ../../include/canal.h ../../include/srplib.h
MPHEADERS = $(HEADERS) ../../include/mp.h
ECPHEADERS = $(HEADERS) ../../include/ecp.h
MPOBJS = mp_m.o mp.o
ECPOBJS = ecp_m.o ecp.o
#
# ------------------ TWORZY WSZYSTKIE PROCESY --------------------------
create: ecp_m mp_m

# ------------------ TWORZY PROCES ECP_M -------------------------------
ecp_m: $(ECPHEADERS) $(ECPOBJS)
   $(LD) $(LDFLAGS) -4 -o $@ $(ECPOBJS) -l $(MRROCPPPATH)/lib/comlib32.lib
#
ecp_m.o: $(ECPHEADERS) ecp_m.cc
   $(WPP) $(WPPFLAGS) ecp_m.cc
#
ecp.o: $(ECPHEADERS) ecp.cc
```

```
   $(WPP) $(WPPFLAGS) ecp.cc
#
# ------------------ TWORZY PROCES MP_M --------------------------------
mp_m: $(MPHEADERS) $(MPOBJS)
   $(LD) $(LDFLAGS) -4 -o $@ $(MPOBJS) -l $(MRROCPPPATH)/lib/comlib32.lib
#
mp_m.o: $(MPHEADERS) mp_m.cc
   $(WPP) $(WPPFLAGS) mp_m.cc
#
mp.o: $(MPHEADERS) mp.cc
   $(WPP) $(WPPFLAGS) mp.cc
#
# Instalacja: kopiuje pliki wykonywalne do katalogu bin
install:
   -@cp -nv ecp_m mp_m ../../bin
#
# Kopiuje z aktualnego katalogu na dyskietkę (zamontowana jako /a) nowsze pliki źródłowe i makefile
save_to_a:
   -@cp -nv *.cc makefile /a/mrrocpp/src/ecp_mp
# -- Kopiuje na dyskietke pliki źródłowe i makefile ze wszystkich
#     podkatalogów mrrocpp
save_all_to_a:
   -@cp -nv *.cc makefile /a/mrrocpp/src/ecp_mp
# EDP
   -@cp -nv ../edp/edp_m.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/edp.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/servo_ms.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/servo_gr.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/hi_rydz.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/rnt_kin.cc /a/mrrocpp/src/edp
   -@cp -nv ../edp/makefile /a/mrrocpp/src/edp
# UI
   -@cp -nv ../ui/ui.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/srp.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/robot.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/teaching.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/moveinc.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/moveint.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/moveext.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/errorinf.cc /a/mrrocpp/src/ui
   -@cp -nv ../ui/makefile   /a/mrrocpp/src/ui
# okienka
   -@cp -nv ../ui/wnd/*.pict /a/mrrocpp/src/ui/wnd
   -@cp -nv ../ui/wnd/*.wnd /a/mrrocpp/src/ui/wnd

# COMM
   -@cp -nv ../comm/canal.cc  /a/mrrocpp/src/comm
   -@cp -nv ../comm/cr_buf.cc /a/mrrocpp/src/comm
   -@cp -nv ../comm/srplib.cc /a/mrrocpp/src/comm
   -@cp -nv ../comm/makefile /a/mrrocpp/src/comm

# pliki *.h
   -@cp -nv ../../include/*.h   /a/mrrocpp/include
   -@cp -nv ../../include/makefile /a/mrrocpp/include

# -- Kopiuje z dyskietki do bieżącego katalogu nowsze pliki zrodlowe i makefile copy_from_a:
   -@cp -nv /a/mrrocpp/src/ecp_mp/*.cc .
   -@cp -nv /a/mrrocpp/src/ecp_mp/makefile .
#
# Kopiuje z dyskietki do podkatalogów wszystkie nowsze pliki źródłowe i makefile
copy_all_from_a:
```

```
   -@cp -nv /a/mrrocpp/src/ecp_mp/*.cc .
   -@cp -nv /a/mrrocpp/src/ecp_mp/makefile .
# EDP
   -@cp -nv /a/mrrocpp/src/edp/edp_m.cc ../edp/edp_m.cc
   -@cp -nv /a/mrrocpp/src/edp/edp.cc ../edp/edp.cc
   -@cp -nv /a/mrrocpp/src/edp/servo_ms.cc ../edp/servo_ms.cc
   -@cp -nv /a/mrrocpp/src/edp/servo_gr.cc ../edp/servo_gr.cc
   -@cp -nv /a/mrrocpp/src/edp/hi_rydz.cc ../edp/hi_rydz.cc
   -@cp -nv /a/mrrocpp/src/edp/rnt_kin.cc ../edp/rnt_kin.cc
   -@cp -nv /a/mrrocpp/src/edp/makefile ../edp/makefile
# UI
   -@cp -nv /a/mrrocpp/src/ui/ui.cc ../ui/ui.cc
   -@cp -nv /a/mrrocpp/src/ui/srp.cc ../ui/srp.cc
   -@cp -nv /a/mrrocpp/src/ui/robot.cc ../ui/robot.cc
   -@cp -nv /a/mrrocpp/src/ui/teaching.cc ../ui/teaching.cc
   -@cp -nv /a/mrrocpp/src/ui/moveinc.cc ../ui/moveinc.cc
   -@cp -nv /a/mrrocpp/src/ui/moveint.cc ../ui/moveint.cc
   -@cp -nv /a/mrrocpp/src/ui/moveext.cc ../ui/moveext.cc
   -@cp -nv /a/mrrocpp/src/ui/errorinf.cc ../ui/errorinf.cc
   -@cp -nv /a/mrrocpp/src/ui/makefile ../ui/makefile

# okienka
   -@cp -nv /a/mrrocpp/src/ui/wnd/*.pict ../ui/wnd
   -@cp -nv /a/mrrocpp/src/ui/wnd/*.wnd ../ui/wnd
# COMM
   -@cp -nv /a/mrrocpp/src/comm/canal.cc ../comm
   -@cp -nv /a/mrrocpp/src/comm/cr_buf.cc ../comm
   -@cp -nv /a/mrrocpp/src/comm/srplib.cc ../comm
   -@cp -nv /a/mrrocpp/src/comm/makefile ../comm
# pliki *.h
   -@cp -nv /a/mrrocpp/include/*.h ../../include
   -@cp -nv /a/mrrocpp/include/makefile ../../include
# Usuwanie plików clean:
   -@rm -v *.o *.err ecp_m mp_m
# kopiuje pliki ECP i MP do uczenia (teach) do plików ecp_m.cc oraz mp_m.cc teach_to_cc:
   -@cp -v ecp_m_t.cc ecp_m.cc
   -@cp -v mp_m_t_c.cc mp_m.cc
   -@rm *.o ecp_m mp_m
#
# kopiuje pliki ECP i MP do kalibracji (calibration) do plików ecp_m.cc oraz mp_m.cc
calib_to_cc:
   -@cp -v ecp_m_c.cc ecp_m.cc
   -@cp -v mp_m_t_c.cc mp_m.cc
   -@rm *.o ecp_m mp_m
#
# kopiuje pliki ECP i MP dla kooperujacych robotów do plików ecp_m.cc oraz mp_m.cc coop_to_cc:
   -@cp -v ecp_m_co.cc ecp_m.cc
   -@cp -v mp_m_co.cc mp_m.cc
   -@rm -v *.o ecp_m mp_m
#
# kopiuje pliki ECP i MP do frezowania do plików ecp_m.cc oraz mp_m.cc frez_to_cc:
   -@cp -v ecp_m_fr.cc ecp_m.cc
   -@cp -v mp_m_t_c.cc mp_m.cc
   -@rm -v *.o ecp_m mp_m
#
# kopiuje pliki ECP i MP z generatorem liniowym do plików ecp_m.cc oraz mp_m.cc linear_to_cc:
   -@cp -v ecp_m_f.cc ecp_m.cc
   -@cp -v mp_m_t_c.cc mp_m.cc
   -@rm -v *.o ecp_m mp_m
#
# kopiuje pliki ECP i MP do identyfikacji do plików ecp_m.cc oraz mp_m.cc ident_to_cc:
```

```
   -@cp -v ecp_m_id.cc ecp_m.cc
   -@cp -v mp_m_t_c.cc mp_m.cc
   -@rm -v *.o ecp_m mp_m
#
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP do uczenia (teach) cc_to_teach:
   -@cp -v ecp_m.cc ecp_m_t.cc
   -@cp -v mp_m.cc mp_m_t_c.cc
#
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP do kalibracji (calibration)
cc_to_calib:
   -@cp -v ecp_m.cc ecp_m_c.cc
   -@cp -v mp_m.cc mp_m_t_c.cc
#
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP dla kooperujacych robotów cc_to_coop:
   -@cp -v ecp_m.cc ecp_m_co.cc
   -@cp -v mp_m.cc mp_m_co.cc
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP do frezowania cc_to_frez:
   -@cp -v ecp_m.cc ecp_m_fr.cc
   -@cp -v mp_m.cc mp_m_t_c.cc
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP generatorem liniowym cc_to_linear:
   -@cp -v ecp_m.cc ecp_m_f.cc
   -@cp -v mp_m.cc mp_m_t_c.cc
#
# kopiuje pliki ecp_m.cc oraz mp_m.cc do plików ECP i MP do identyfikacji cc_to_ident:
   -@cp -v ecp_m.cc ecp_m_id.cc
   -@cp -v mp_m.cc mp_m_t_c.cc
# help:
   -@echo 'create          - tworzy wszystkie procesy (domyslne)'
   -@echo 'install         - kopiuje pliki wykonywalne ecp_m i mp_m do
                             katalogu ../mrrocpp/bin'
   -@echo 'save_to_a       - kopiuje pliki *.cc i makefile na dyskietke'
   -@echo 'save_all_to_a   - kopiuje pliki *.cc i makefile ze wszystkich
                             podkatalgów na dyskietke'
   -@echo 'copy_from_a     - kopiuje pliki *.cc i makefile z dyskietki'
   -@echo 'copy_all_from_a - kopiuje pliki *.cc i makefile z dyskietki
                             do podkatalogów'
   -@echo 'clean           - usuwa pliki wykonywalne, pliki *.o *.err'
   -@echo ' '
   -@echo 'Opcje do tworzenia specyficznych sterowników:'
   -@echo 'cc_to_teach  - kopiowanie mp_m.cc i ecp_m.cc na mp_m_t_c.cc ecp_m_t.cc'
   -@echo 'cc_to_calib  - kopiowanie mp_m.cc i ecp_m.cc na mp_m_t_c.cc ecp_m_c.cc'
   -@echo 'cc_to_coop   - kopiowanie mp_m.cc i ecp_m.cc na mp_m_co.cc ecp_m_co.cc'
   -@echo 'cc_to_frez   - kopiowanie mp_m.cc i ecp_m.cc na mp_m_t_c.cc ecp_m_fr.cc'
   -@echo 'cc_to_ident  - kopiowanie mp_m.cc i ecp_m.cc na mp_m_t_c.cc ecp_m_id.cc'
#
   -@echo 'teach_to_cc  - kopiowanie mp_m_t_c.cc ecp_m_t.cc na mp_m.cc i ecp_m.cc'
   -@echo 'calib_to_cc  - kopiowanie mp_m_t_c.cc ecp_m_c.cc na mp_m.cc i ecp_m.cc'
   -@echo 'coop_to_cc   - kopiowanie mp_m_co.cc ecp_m_co.cc na mp_m.cc i ecp_m.cc'
   -@echo 'frez_to_cc   - kopiowanie mp_m_t_c.cc ecp_m_fr.cc na mp_m.cc i ecp_m.cc'
   -@echo 'ident_to_cc  - kopiowanie mp_m_t_c.cc ecp_m_id.cc na mp_m.cc i ecp_m.cc'
```

Wywołanie `make` bez argumentów (domyślnym argumentem jest `-f makefile`) powoduje kompilację i konsolidację programów składowych procesów ECP i MP. Po ich prawidłowym przebiegu należy pamiętać o skopiowaniu nowych plików wykonywalnych do katalogu `../mrrocpp/bin`. Najprościej można to wykonać wywołując `make install`. Dla uzyskania pomocy o innych opcjach należy uruchomić `make help`. Pliki `makefile` można modyfikować stosownie do potrzeb.

Literatura

[1] Zieliński C., Szynkiewicz W.: System MRROC++ dla robota IRp-6 na torze jezdnym, Instytut Automatyki i Informatyki Stosowanej, raport wewnętrzny IAIS nr 99-20, 1999.

[2] Zieliński C., Szynkiewicz W.: Podręcznik programowania systemu MRROC++ dla robota IRp-6 na torze jezdnym, Instytut Automatyki i Informatyki Stosowanej, raport wewnętrzny IAIS nr 99-21, 1999.