Politechnika Łódzka
Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki
Instytut Elektroenergetyki

PRACA DYPLOMOWA MAGISTERSKA

Konfiguracja systemu czasu rzeczywistego na bazie Xenomai/RTAI

Real time system based on Xenomai/RTAI

Mateusz Fraszczyński
192299

Opiekun pracy
dr hab. inż. Grzegorz Granosik
Dodatkowy opiekun pracy
mgr. inż. Marek Gawryszewski

Łódź, wrzesień 2016

Serdeczne podziękowania dla dr hab. inż. Grzegorza Granosika, prof. PŁ oraz mgr. inż. Marka Gawryszewskiego za okazaną pomoc, czas i życzliwość.

Spis treści

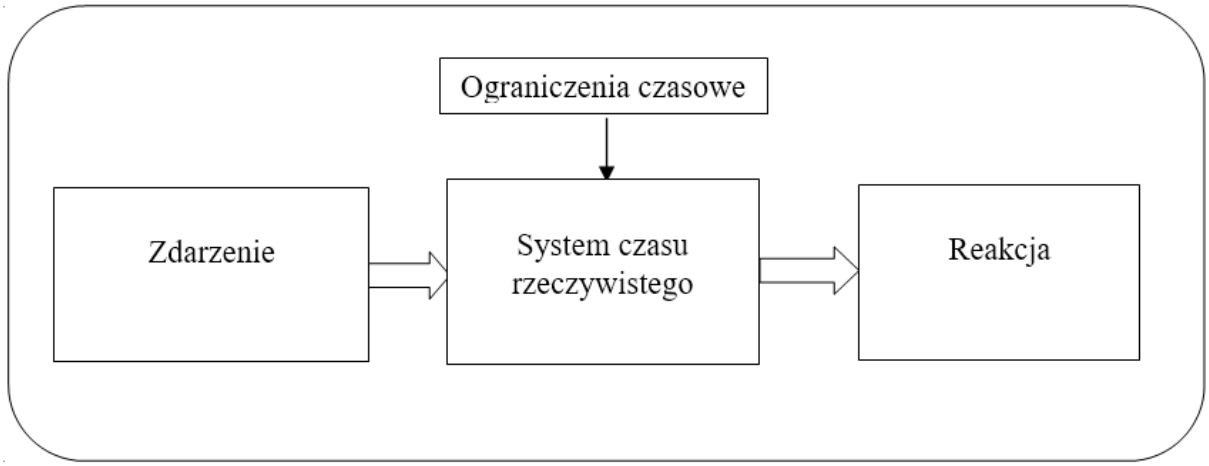
Rozdział 1.

Cel i zakres pracy

Systemy operacyjne są powszechnie używane w dzisiejszych komputerach, zarówno tych klasy PC, jak i mniejszych urządzeniach wykorzystywanych w przemyśle. Coraz nowsze systemy przemysłowe sprawiają, że wymogi stawiane systemom operacyjnym obsługującym te układy są z roku na rok wyższe.

Precyzja układów przemysłowych rośnie, wiąże się z tym potrzeba stosowania stabilniejszych oraz szybszych systemów, które będą niezawodne, jednocześnie zachowując przy tym szybkość działania. W związku z tym klasyczne systemy operacyjne stosowane powszechnie w sektorze prywatnym nie są wystarczające aby spełnić te wymogi i z czasem zaczęto stosować dedykowane systemy operacyjne do danych czynności. Ze względu na otwarty dostęp coraz większą popularność zyskuje system Linux, który poprzez modyfikacje dostosowywany jest do dedykowanych systemów.

Wymogi stawiane współczesnym systemom wiążą się z szybkimi odpowiedziami na sygnały przychodzące z zewnątrz oraz odpowiednią reakcją na nie w ściśle określonym czasie, z tego względu projektowanie takich systemów jest bardzo złożonym procesem. W klasycznym ujęciu każdy system, mający na celu odpowiednio zareagować na zdarzenie zewnętrzne w określonym czasie nazywany jest systemem czasu rzeczywistego *(Rysunek 1*). Nie istnieje ściśle określona granica czasu reakcji, po którym można uznać dany system za czasu-rzeczywistego, jednak ważne jest, że powinien zareagować we wcześniej narzuconych ramach czasowych.



Rysunek 1. Schemat systemu czasu rzeczywistego

Celem pracy było przygotowanie i przetestowanie takiego systemu, którego działanie zawierałoby się w określonych ograniczeniach czasowych. W tym celu stworzony został system na bazie Linuxa ze zmienionym jądrem na takie, które będzie spełniać wymogi systemu czasu rzeczywistego. Do testowania systemu posłużyło programowalne urządzenie, na którym można uruchomić stworzony system – jest nim moduł Raspberry PI.

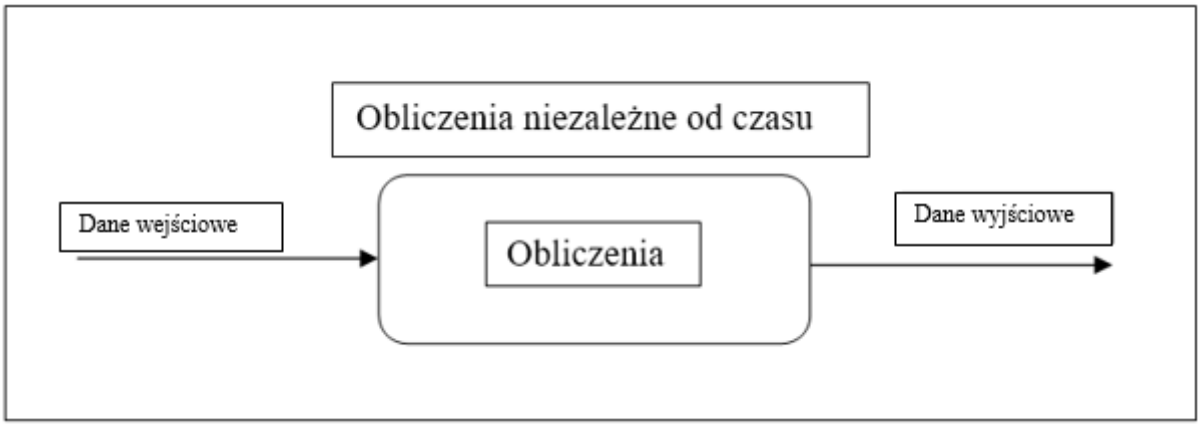
Rozdział 2.

Systemy czasu rzeczywistego

2.1 Definicja systemu czasu rzeczywistego

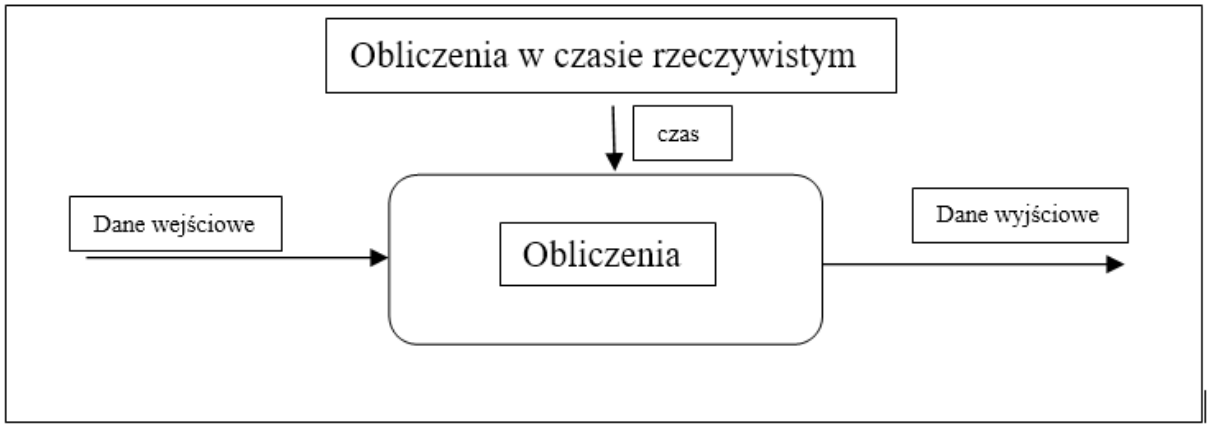
Istnieje wiele definicji systemów RTOS[1], nie mniej jednak wszystkie z nich związane są z narzuconym czasem na działanie systemu. Najpopularniejszą definicją takiego systemu jest:

„System czasu rzeczywistego (ang. Real Time System) jest to system komputerowy, w którym obliczenia są wykonywane współbieżnie z procesami zewnętrznymi (z otoczeniem) w celu sterowania, nadzorowania i terminowego reagowania na zdarzenia występujące w tych procesach. Cechą charakterystyczną systemów czasu rzeczywistego jest ścisłe sprzężenie pomiędzy otoczeniem, w którym przebiegają procesy rzeczywiste, a systemem, który musi te zdarzenia identyfikować i reagować na nie w ściśle określonym czasie”[2]



Rysunek 2.1 Uproszczony model ogólnego systemu [1]

[1] Real Time Operating System (RTOS) – System czasu rzeczywistego
[2] Paweł Majdzik Programowanie współbieżne



Rysunek 3.2 Uproszczony model systemu czasu rzeczywistego [1]

Często RTOS definiuje się jako system, w którym można określić maksymalny czas wykonania poszczególnych operacji, wynika z tego, że poprawność działania takiego systemu nie zależy tylko i wyłącznie od samego wyniku danej operacji ale także od tego w jakim czasie został on osiągnięty.

2.2 Podział systemów czasu rzeczywistego

Ogólnie systemy czasu rzeczywistego możemy podzielić na trzy kategorię:

- System o twardych wymaganiach czasowych (Hard Real-Time Systems). W takim systemie przekroczenie ostrych ograniczeń czasowych powoduje poważne skutki, a niekiedy nawet katastrofalne. Przykładem tego typu systemów mogą być: systemy kontroli lotu, systemy wojskowe, sterowanie elektrowni atomowej.
- Systemy o mocnych wymaganiach czasowych (ang. Firm Real-Time Systems) W tego typu systemach przekroczenie limitu czasowego może powodować utratę przydatności tych obliczeń, a przekroczenie limitów może powodować

dodatkowe koszty, lecz nie powoduje zagrożenia życia lub zdrowia. Przykładem takiego systemu mogą być: systemy zleceń giełdowych, obsługa bankowości.

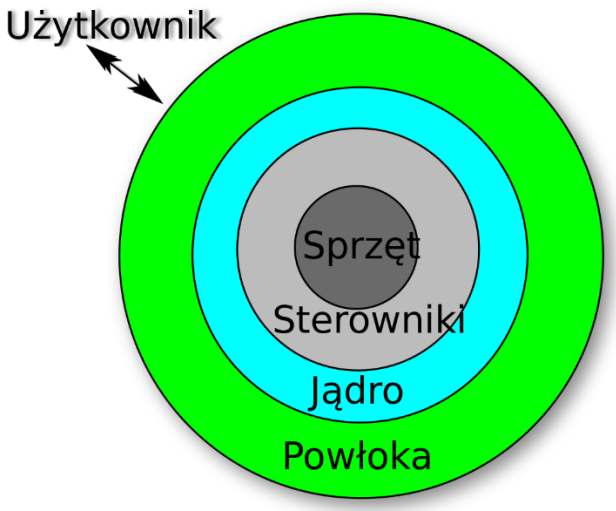
- Systemy o miękkich wymaganiach czasowych (ang. Soft Real-Time Systems) Są to systemy, w których przekroczenie nałożonych terminów czasowych nie powoduje zagrożenia ani strat, a jedynie zmniejsza przydatność takiego systemu, poprzez obniżenie poprawności jego działania. Przykładem takiego systemu może być: bankomat, nawigacja samochodowa.

2.3 Zastosowanie

Obecnie systemy RTOS są powszechnie stosowane w wielu dziedzinach, głównym obszarem ich zastosowań są technologie wojskowe, telekomunikacyjne, sterowanie elektroniką samochodową (np. ABS, poduszki powietrzne), maszyny przemysłowe (roboty). Jednakże systemy te znajdują także zastosowanie w bankomatach, elektronicznych bilbordach czy inteligentnych domach. Praktycznie system taki da się wykorzystać wszędzie, a wszystko zależy od narzuconych ram czasowych.

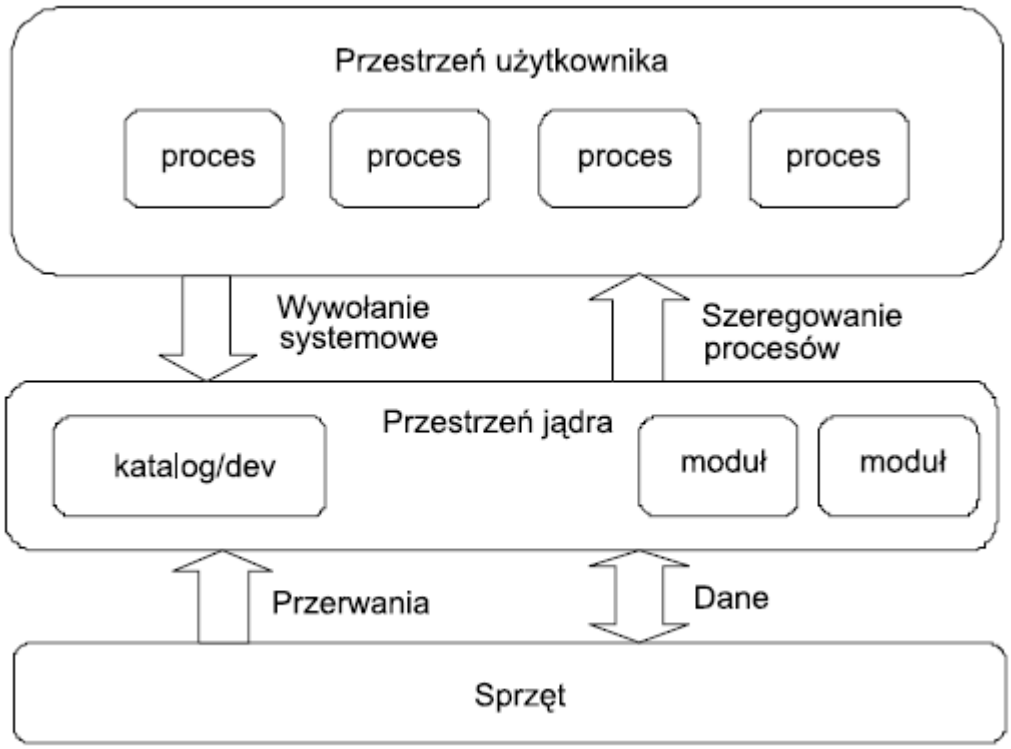
2.4 Xenomai

Xenomai jest darmowym oprogramowaniem wspierającym działanie jądra Linux. W przypadku gdy samo jądro systemu nie jest w stanie spełnić wymogów czasowych istnieje możliwość stworzenie jądra Xenomai, które będzie współpracować z jądrem Linuxa pozwalając na osiągnięcie limitów czasowych (*Rysunek 3*).



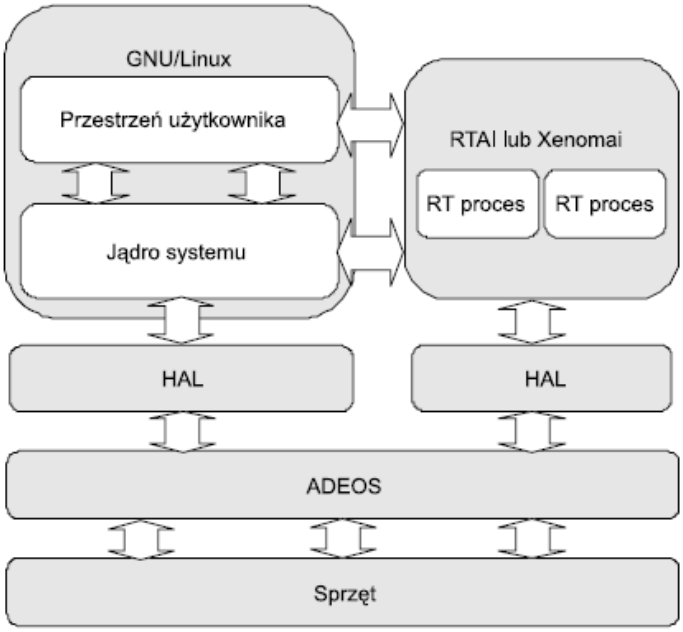
Rysunek 3. Schemat powłok systemu [8]

Xenomai tworzone jest głównie z myślą o systemach wbudowanych, nie mniej jednak możliwe jest jego użycie do aplikacji desktopowych. W systemach zbudowanych z Xenomai tworzone jest współegzystujące z jądrem Linuxowym, jądro o nazwie Cobalt. Jądro to przejmuje wszystkie czasowo krytyczne zadania jakimi mogą być: obsługa przerwań, przełączanie (scheduling) wątków. Jądro to posiada wyższy priorytet działania od standardowego jądra Linuxa. Aby móc zarządzać zadaniami oraz procesami wewnątrz jądra Cobalt Xenomai dostarcza dedykowane do tego API (interfejs programistyczny aplikacji). W klasycznych systemach Linux z pojedynczym jądrem wszystkie zadnia realizowane są w jednej z przestrzeni: jądra bądź użytkownika (*Rysunek 4.1*). Przy czym wszelkie procesy wykonywane są w przestrzeni użytkownika, zaś moduły systemowe wykonywane są w przestrzeni jądra. Zarządzać procesami można poprzez nadawanie im priorytetów, lecz i to nie gwarantuje, że wykonane zostaną w określonym czasie, gdyż moduły systemu mają pierwszeństwo.



Rysunek 4.1 Architektura systemu GNU/Linux [4]

Wykorzystanie mikro jądra Xenomai pozwala na przeniesienie części procesów z przestrzeni użytkowania do przestrzeni jądra Xenomai, które współpracuje bezpośrednio z jądrem systemu. Pozwala to na szybszą realizację procesów z większym priorytetem. Dodatkowo wszelkie procesy czasu rzeczywistego mogą komunikować się ze zwykłymi procesami oraz innymi procesami czasu rzeczywistego, co schematycznie pokazano na *Rysunku 4.2*.



Rysunek 4.2 Architektura systemu GNU/Linux wraz z jądrem Xenomai/RTAI [4]

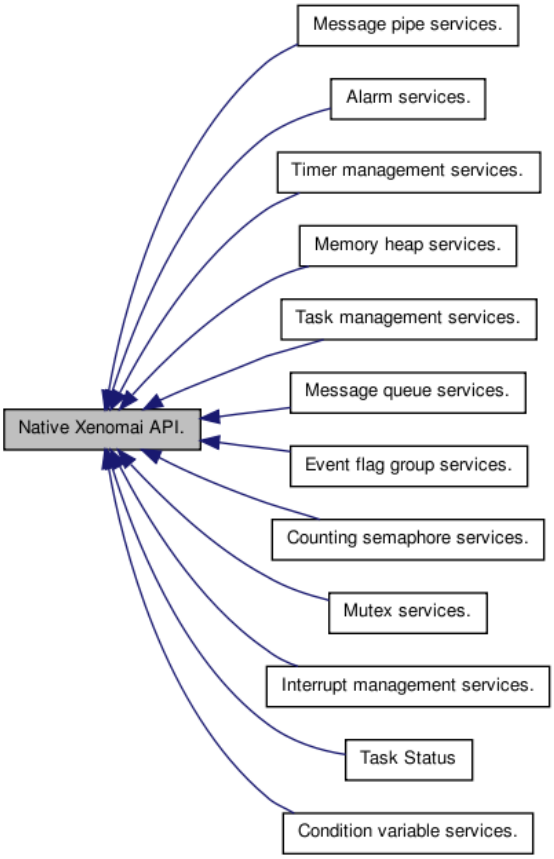
Procesy z zastosowaniem jądra Xenomai mogą być wywoływane zarówno z poziomu przestrzeni użytkownika, jak i przestrzeni jądra. Najprostsze użycie procesów czasu rzeczywistego odbywa się z poziomu przestrzeni użytkownika, lecz daje gorsze rezultaty czasowe niż wywołanie z przestrzeni jądra, które wymaga odpowiedniej nietrywialnej implementacji.

Xenomai wraz z jądrem czasu rzeczywistego dostarcza także interfejs umożliwiający pełne wykorzystanie możliwości jakie daje nowe jądro. Xenomai API składa się z kilku modułów, które mogą być wykorzystane:

- Xenomai nucleus
- HAL
- Native Xenomai API
- Real-Time Driver Model
- Analogy API
- POSIX skin

Do standardowych aplikacji niewymagających ścisłej współpracy ze sprzętem (hardwarem) stosowany jest interfejs Native Xenomai API. Interfejsy takie jak Xenomai nucleus, HAL[3], czy Real-Time Driver Model stosowane są przy tworzeniu zadań z poziomu przestrzeni jądra. Do tworzenia zadań z przestrzeni użytkownika wystarczą Native Xenomai API oraz POSIX skin.

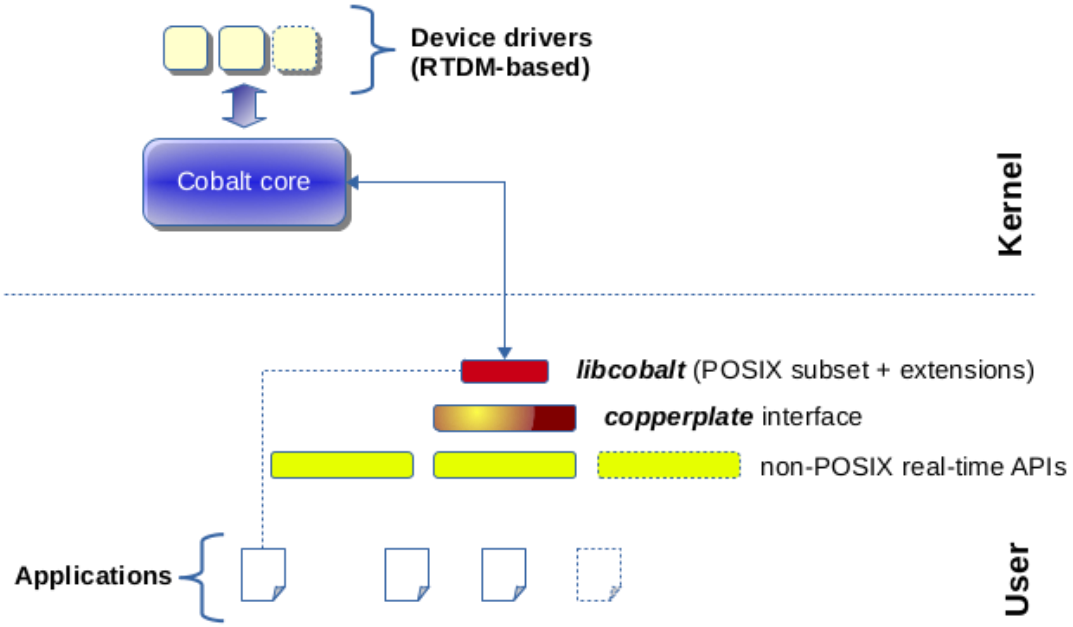
W ramach tej pracy obszar używania Xenomai API został zawężony do Native Xenomai API, ze względu na jego największą użyteczność dla tworzonej aplikacji testowej (*Rysunek 5*). Na Native Xenomai API składa się szereg serwisów, które mogą być wykorzystane w projekcie, np.: Alarm service, Timer management service czy Task management service, który jest niezbędny do utworzenia zadania czasu rzeczywistego.



Rysunek 5. Native Xenomai API [9]

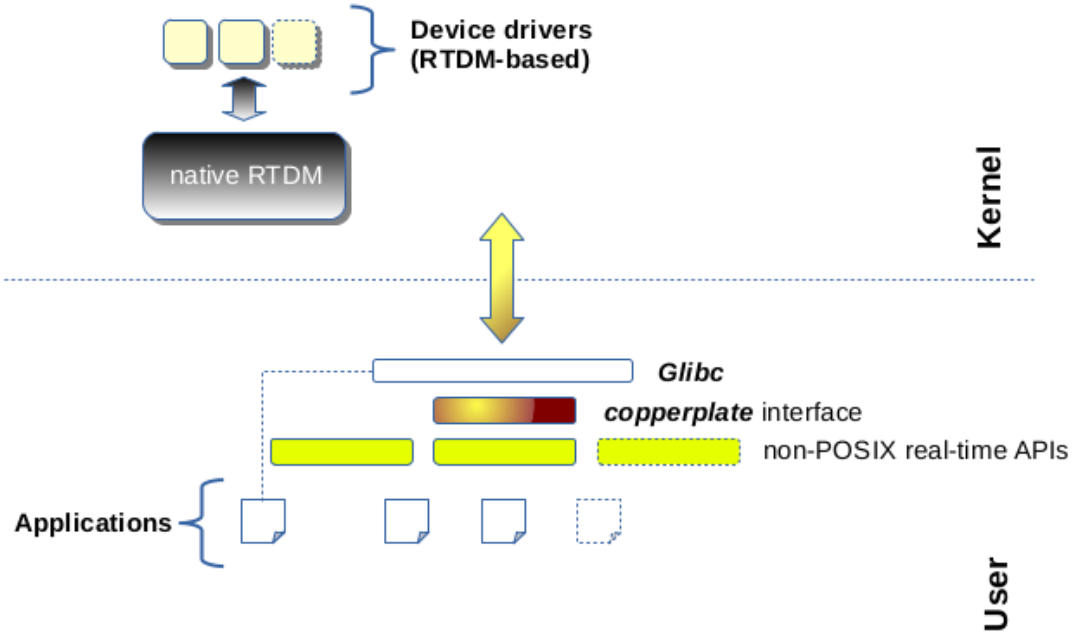
[3] HAL – Hardware abstract layer

Najnowsze Xenomai w wersji 3 wprowadza nową architekturę, oprócz wspieranego w wersji 2 mikro jądra Cobalt istniej możliwość wykorzystania pojedynczego jądra bazującego na możliwościach standardowego jądra Linuxa, tworząc w ten sposób jądro zwane Mercury. Dzięki zastosowaniu nakładki PREEMPT_RT do standardowego jądra Linux, używając Xenomai 3 można uzyskać jeszcze krótsze czasy reakcji, niż stosując wcześniejsze wersje z podwójnym jądrem. Pozwala to także na uproszczenie aplikacji, które nie wymagają tworzenia osobnych zadań czasu rzeczywistego.



Rysunek 6.1 Architektura podwójnego jądra Cobalt [7]

Zastosowany w Xenomai 3 RTDM (Real Time Driver Model) dostarcza ujednolicony interfejs zarówno dla użytkownika, jak i developera, co znacznie przyspiesza działanie aplikacji oraz ułatwia jej obsługę. Xenomai 3 jest tak zaprojektowane, że możliwe jest uruchamianie aplikacji zarówno z pojedynczym jądrem, jak i podwójnym bez konieczności zmian w implementacji.



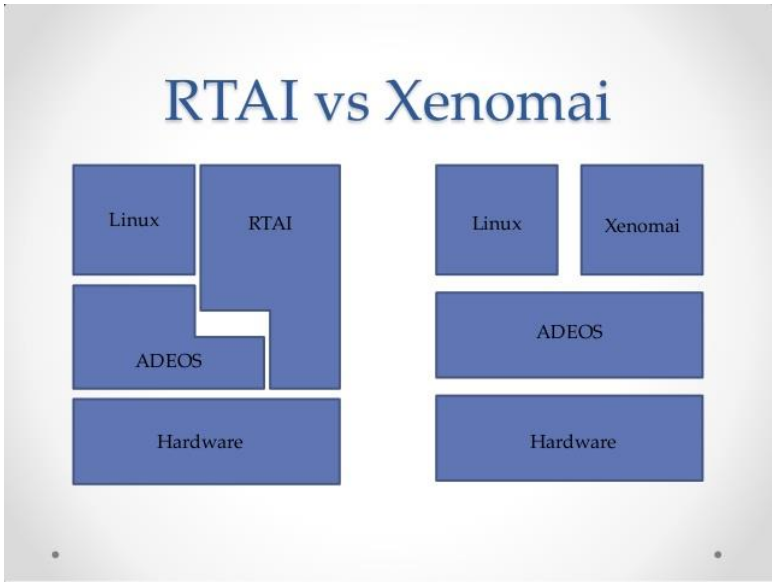
Rysunek 6.2 Architektura pojedynczego jądra Mercury [7]

2.5 RTAI

RTAI (Real Time Application Interface) jest systemem tworzonym z myślą przede wszystkim o wydajności czasowej. To odróżnia go nieco od Xenomai, które stawia na wygodny i praktyczny interfejs użytkownika, który pozwala na łatwe przeportowanie aplikacji na nowe jądro. RTAI jest także oprogramowaniem otwartego dostępu (open-source). Działanie tego systemu polega na ładowaniu do jądra dedykowanych modułów, pozwalających na tworzenie zadań o twardych ograniczeniach czasowych. Podobnie jak Xenomai 2 system ten tworzy mikro jądro współpracujące ściśle z jądrem Linuxa. Moduły dostępne dla RTAI:

- Moduł *rtai* – jest podstawowym i najważniejszym modułem, odpowiedzialnym za inicjalizacje i zarządzenie procedurami obsługi przerwań, komunikację z jądrem Linux;
- Moduł *rtai_sched* – w tym module znajduje się scheduler odpowiedzialny za zarządzenia zadaniami wewnątrz systemu. Moduł ten na bazie priorytetów przydziela zadaniom czas procesora;

- Moduł *rtai_fifos* – przy pomocy tego modułu możliwe jest wykorzystanie kolejek FIFO (first in, first out);
- Moduł *rtai_shm* – dzięki temu modułowi możliwe jest dzielenie pamięci pomiędzy procesy RT a zwykłe Linuxowe;
- Moduł *rtai_lxrt* – ten moduł pozwala na komunikację pomiędzy standardowymi zadaniami Linuxowymi a RT.



Rysunek 7. Porównanie RTAI i Xenomai [13]

Rozdział 3.

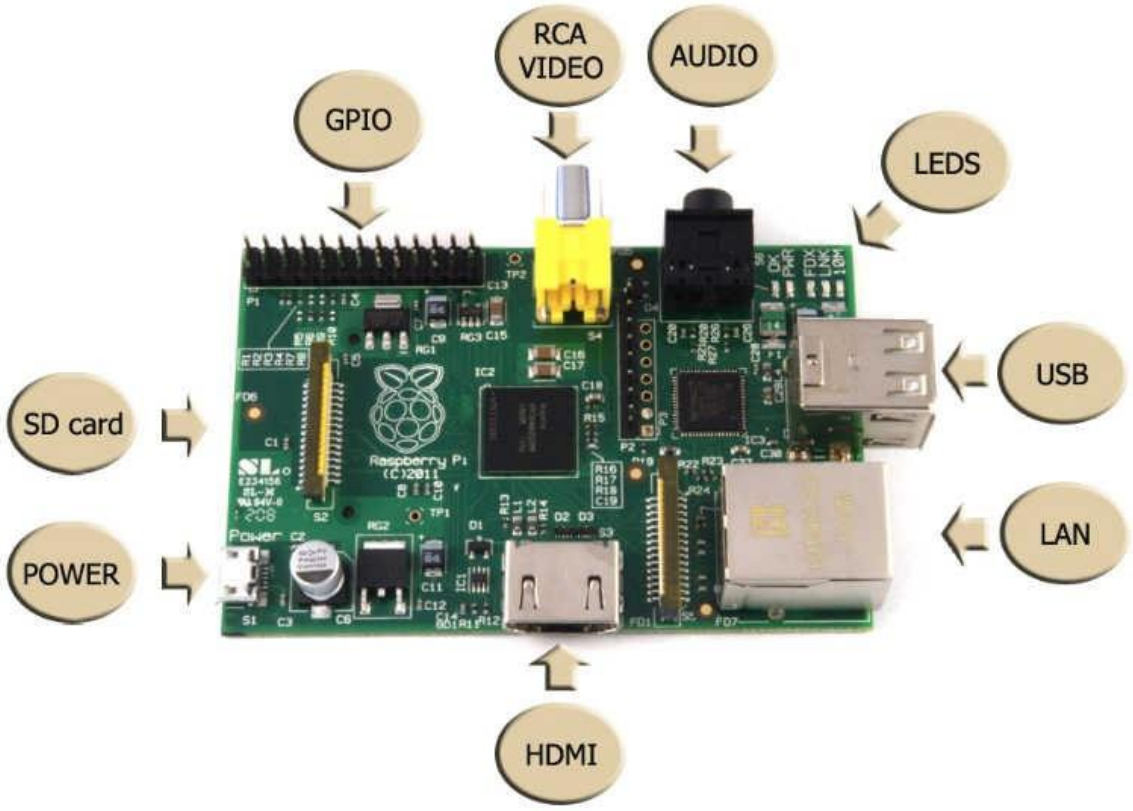
Raspberry PI

Każdy system operacyjny musi być uruchamiany na fizycznym urządzeniu, którym może być komputer stacjonarny klasy PC bądź też dedykowany system wbudowany posiadający procesor. Programowalne systemy wbudowane musza wykonywać zadane operacje na sterowanym przez nie sprzęcie, niekiedy jednak posiadają zbyt słabe procesory aby móc na nich uruchomić system czasu rzeczywistego. Z kolei konfiguracja RTOS na sprzęcie klasy PC posiada szereg ograniczeń w postaci wielu zadań jakie procesor wykonuje, obsługując cały sprzęt. W takim systemie trudno jest osiągnąć pełną kontrolę nad jego działaniem. W związku z czym w ramach niniejszego projektu zdecydowano się wybrać sprzęt, który posiada cechy zarówno komputera klasy PC, jak i systemów wbudowanych. Raspberry PI jest małym przenośnym komputerem, który można programować zarówno bez wsparcia jakiegokolwiek systemu, jak i wykorzystując system operacyjny. Sprzęt ten jest niezwykle uniwersalny i pozwala na tworzenie zarówno prostych, jak i bardzo skomplikowanych projektów.

3.1 Opis urządzenia

Raspberry PI występuje w wielu wersjach i modelach. Pierwszym jaki powstał był model A posiadający procesor 700 MHz ARM1176JZF-S oraz pamięć SDRAM 256 MB. Na płytce drukowanej znajduje się także wejście USB, Video oraz Audio. Dodatkowo Rasberry PI posiada wyjście HDMI. Nośnikiem danych w tym systemie jest karta SD. Zastosowany w ramach pracy Model B, odróżnia się od wyżej opisanego modelu tym, że posiada dodatkowy port USB oraz port sieciowy Ethernet. Modele nowsze od zastosowanego w ramach projektu różnią się głównie procesorem. Od modelu B2 zaczęto stosować procesor 900 MHz quad-core ARM cortex-A7, który jest znacznie wydajniejszy od swojego poprzednika. Zaletą Raspberry

PI jest także wyprowadzony szereg portów GPIO, które mogą być wykorzystane do różnych celów.



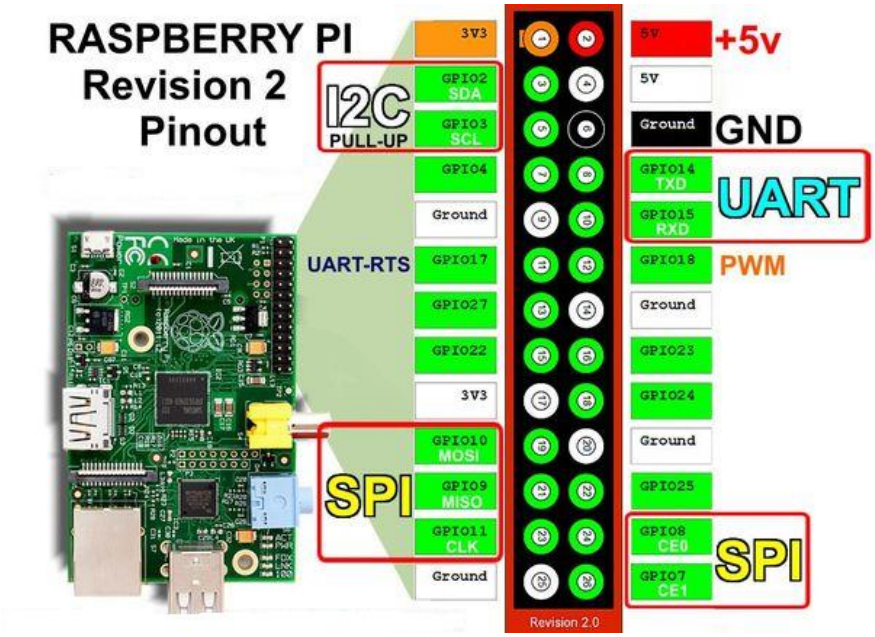
Rysunek 8. Raspberry PI Model B [14]

Część portów GPIO ma swoje dedykowane zadania jak np. komunikacja I2C lub UART. Porty te mogą pracować w określonym trybie bądź też mogą być sterowane przez aplikacje programu w dowolny sposób.

Do przetestowania działania systemu operacyjnego czasu rzeczywistego przy pomocy Raspberry PI wykorzystane zostało generowanie fali PWM[4]. Ponieważ sprzęt dostarcza wbudowany moduł do obsługi PWM działający niezależnie od procesora pozwoli to na porównanie jego pracy z wynikami osiąganymi dla programowego PWM, który będzie działać

[4] PWM (Pulse-Width Modulation) – Generowana fala prostokątna o zadanym wypełnieniu

w ramach systemu operacyjnego. Sprzętowy PWM znajduję się na GPIO18 czyli pinie o numerze 12, zaś programowy PWM może działać na dowolnym pinie.



Rysunek 9. Raspberry PI Pinout [15]

Ponieważ generowanie fali prostokątnego wiąże się z zadawaniem częstotliwości oraz wypełnienia, pozwoli to na sprawdzenie jak system radzi sobie ze zmianą wartości sygnału na pinie przy dużych częstotliwościach, a co za tym idzie bardzo krótkich czasach reakcji. Innymi słowy zadaniem systemu będzie sterowanie pinem w ściśle określonych chwilach czasowych.

3.2 System operacyjny

Raspberry PI wraz ze sprzętem dostarcza też szereg systemów operacyjnych otwartego dostępu (open-source) dedykowanych do tego sprzętu. Większość tych systemów opartych jest na bazie systemu Linux. Systemy działające na Raspberry PI:

- Raspbian
- Debian GNU/Linux
- Ach Linux ARM
- RISC OS

- AROS
- Android 4.0
- FreeBSD
- Gentoo Linux
- Google Chrome OS
- Slackware ARM

Praca z dowolnym system jest bardzo łatwa, wystarczy pobrać jego darmową wersję a następnie wgrać na kartę SD, którą następnie należy umieścić w urządzeniu.

W ramach projektu wykorzystany został system Raspbian w wersji wheezy (*16.02.2015*), gdyż jest on rekomendowanym systemem przez producenta oraz można dla niego zastosować nakładkę Xenomai.

3.3 Aplikacja testująca

W celu przetestowania działania systemu stworzona została aplikacja pozwalająca na wygenerowanie fali PWM na dwa sposoby: poprzez wsparcie hardwarowe oraz softwarowo. Oprócz wspomnianego PWM, w ramach programu przygotowane zostały także zadania mające na celu obciążenie procesora czasochłonnymi obliczeniami. Wszystkie zadania zostały utworzone przy pomocy bibliotek POSIX, która jest charakterystycznym elementem składowym systemów z rodziny Unix, którego częścią jest Linux. Biblioteka ta pozwala na tworzenie między innymi wątków, ich synchronizację, mutexy itp. Tworzenie wielowątkowego systemu przy użyciu POSIX nie zapewnia działania systemu jako czasu rzeczywistego, pozwala jedynie na zaimplementowanie wielowątkowości.

Aplikacja została napisana przy pomocy języka C++, a wszystkie zadania dla procesora stworzone zostały przy pomocy pthread_create z najwyższym priorytetem działania.

3.3.1 Hard-PWM

Do wykorzystania sprzętowego PWM w ramach aplikacji posłużyła biblioteka otwartego dostępu (autorstwa Hussama al-Hertani) pozwalająca na zainicjalizowanie i skonfigurowanie odpowiednio pinu odpowiedzialnego za PWM oraz modułu peryferyjnego sterującego nim. Biblioteka ta dostępna jest na stronie http://hertaville.com pod nazwą `rpiPWM1`. Uruchomienie PWM sprowadzało się dzięki tej bibliotece do napisania prostej metody:

```
void *pwmThread(void *thread_id) {
	rpiPWM1 pwm(1000.0, 256, 80.0, rpiPWM1::MSMODE);
	double dcycle = 0.0;
	while(dcycle < 99.99) {
		pwm.setDutyCycle(dcycle);
		dcycle += 1;
		usleep(200);
		getStamp("PWM");
	}
	endThread(thread_id);
	pthread_exit((void*)thread_id);
}
```

Przy tworzeniu obiektu klasy w parametrach podaje się częstotliwość fali oraz jej wypełnienie.

3.3.2 Soft-PWM

Kod programu do sterowania sprzętowym PWM został stworzony na bazie powszechnie dostępnych przykładów sterowania GPIO w Raspberry PI. Wątek odpowiedzialny za sterowanie softwarowym PWM wygląda następująco:

```
void *gpioThread(void *thread_id) {
	gpioC test(7);
	if(test.gpioPWMInit(measurePtr, 10, 50)) {
		test.gpioPWMStart();
	}

	endThread(thread_id);
	pthread_exit((void*)thread_id);
}
```

Jak widać w powyższym kodzie, do tego celu wykorzystany został pin 7, który przy tworzeniu obiektu klasy został odpowiednio skonfigurowany tak aby można było zmieniać jego wartość. Inicjalizacja PWM polega na podaniu trzech parametrów jakimi są:

- measurePtr – służy do zapisu stempli czasowych, w których następowała zmiana stanu pinu. W późniejszym etapie badań pobór próbek został wyłączony ze względu na zajmowany czas procesora;
- częstotliwość – w tym parametrze podawana jest wartość częstotliwości z jaką PWM ma pracować;
- wypełnienie – ostatni parametr odpowiedzialny jest za podanie wartości wypełnienia fali.

Po podaniu tych parametrów w tworzonym obiekcie klasy jej pola zostawały nadpisywane tymi wartościami. Następnie uruchomienie PWM polega na wywołaniu odpowiedniej metody klasy odpowiedzialnej za wystartowanie PWM.

```
void gpioC::gpioPWMStart() {
	unsigned int periodCount = 0;
	double period = 1 / freq;
	double timeOn  = (( period * duty ) / 100) * 1000000;
	double timeOff = (( period * (100 - duty)) / 100) * 1000000;

	measurePtr->getTimeStamp("SoftPWM_Measure_START");

	cout << "DEbug timeon " << timeOn << " us"<< endl;
	gpioExportPin(this->pin);

	while(periodCount < 100) {

		gpioSet();
		while(!gpioReadPin(this->pin));
		measurePtr->getTimeStamp("SoftPWM_PIN_ON ");
		usleep(timeOn);
		gpioClear();
		while(gpioReadPin(this->pin));
		measurePtr->getTimeStamp("SoftPWM_PIN_OFF ");
		usleep(timeOff);
		periodCount++;
	}
	gpioUnExportPin(this->pin);
}
```

Jak widać na powyższym przykładzie, na podstawie podanych wartości częstotliwości i wypełnienia obliczony został czas jaki pin znajduje się w stanie wysokim, tj. 5V (`timeOn`) i czas w jakim wartość napięcia na pinie wynosi 0V (`timeOff`). W pętli `while`, która wykonuje się 100-krotnie, a każdy jej przebieg odpowiada okresowi fali wynikającemu z częstotliwości, wartość pinu zostawała odpowiednio zmieniana i dodatkowo pobierany stempel czasowy. Jak już wspomniano, w późniejszej fazie testowania pobieranie stempli czasowych zostało wyłączone a ilość przebiegów pętli zwiększona.

3.3.3 Wątki obciążające procesor

Na potrzeby badań stworzone zostały także wątki mające na celu maksymalne obciążenie procesora obliczeniami, aby sprawdzić jak system sobie radzi z wieloma zadaniami. W tym celu powstały wątki odpowiedzialne za:

- Sortowanie bąbelkowe
- Sortowanie poprzez wstawianie
- Obliczanie permutacji tablicy
- Obliczenia trygonometryczne

Na potrzeby sortowań stworzona została klasa, odpowiedzialna za dany rodzaj sortowania, zaś na potrzeby obliczeń matematycznych również osobna klasa do tego celu. Wątki obciążające zostały zaprojektowane tak, aby na potrzeby testów łatwo można było je włączać i wyłączać.

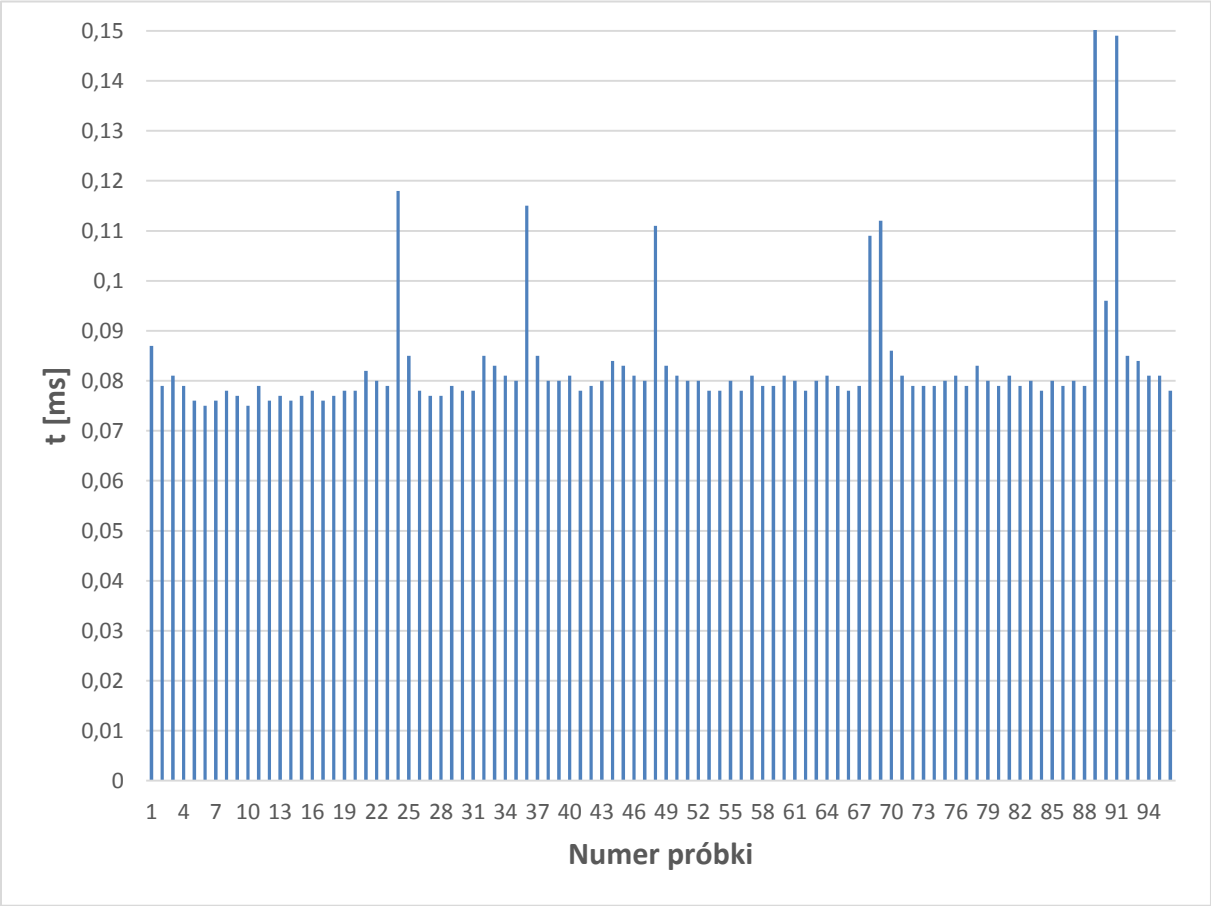
3.4 Pomiary

Pomiary zostały podzielone na dwa etapy. Pierwszy etap polegał na zbieraniu stempli czasowych w chwilach zmian wartości napięcia na pinie. Pomiary zostały wykonane dla różnych wartości częstotliwości, aby sprawdzić czas obliczeń jakie procesor potrzebuje na wykonanie zadania. Dodatkowym czynnikiem mającym wpływ na wyniki pomiarów było manipulowanie wątkami obciążającymi procesor, dla niektórych pomiarów był one włączone, dla innych nie. Drugi etap polegał na mierzeniu sygnału na wyjściu pinu przy pomocy oscyloskopu. Na podstawie przebiegu sygnału wyjściowego mierzona była wartość częstotliwości w stosunku do wartości zadanej.

3.4.1 Stemple czasowe

W pierwszej kolejności uruchomiony został tylko softwarowy PWM, bez jakichkolwiek wątków obciążających z najwyższym możliwym priorytetem wątku. W czasie działania kodu programu przy każdorazowej zmianie wartości napięcia na pinie pobierana była próbka czasowa. Proces ten polega na startowaniu licznika czasu, który przy każdorazowym zapisie próbki zostawał zerowany. Pomiar ten odbywał się dla różnych wartości częstotliwości.

- Częstotliwość = 10000 Hz; Okres = 0,1 ms; Wypełnienie 50%; Czas włączenia (5V) 0,05 ms; Czas wyłączenia (0V) 0,05 ms; Liczba próbek 100.

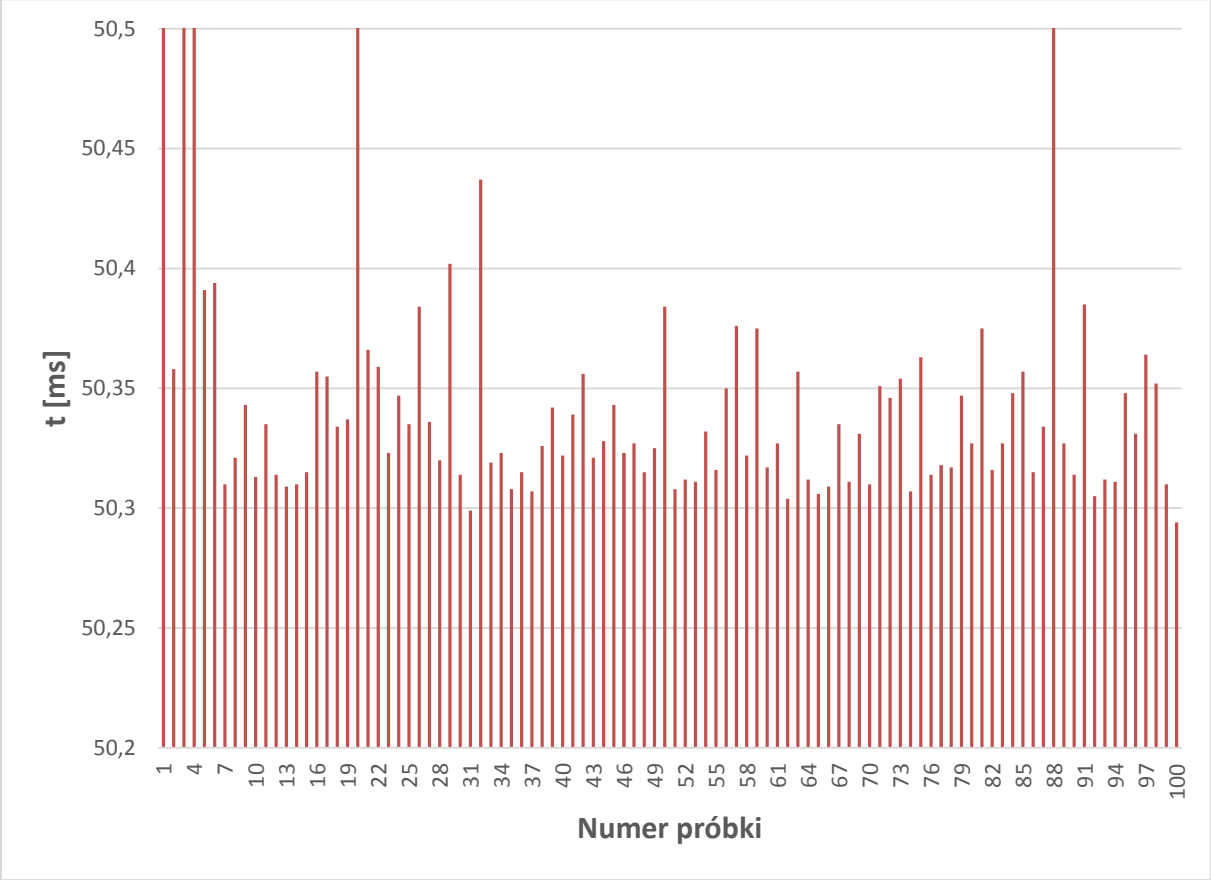


Wykres 1. T_z = 0,1 ms, $T_{on/off}$ = 0,05 ms

Jak widać na *Wykresie 1*. każda zmiana wartości pinu, która powinna następować po 0,05 ms następowała z opóźnieniem. Niekiedy próbki bywały znacznie opóźnione w stosunku do wartości zadanej, co mogło być spowodowane przez inne operacje systemowe jakie procesor musiał przetworzyć w danym momencie. Średnia wartość pomiaru wyniosła 0,1057 ms, co oznacza, że każda zmiana wartości następowała z ponad dwukrotnym opóźnieniem. Jeden pomiar wychodzi poza skale wykresu i ma on wartość 2,33 ms, dla większej czytelności wykresu został on pozostawiony poza skalą.

$$Delay = \left(\frac{0{,}1057 - 0{,}05}{0{,}05} * 100\%\right) = 111{,}4\ \%$$

- Częstotliwość = 10 Hz; Okres = 100 ms; Wypełnienie 50%; Czas włączenia (5V) 50 ms; Czas wyłączenia (0V) 50 ms; Liczba próbek 100.



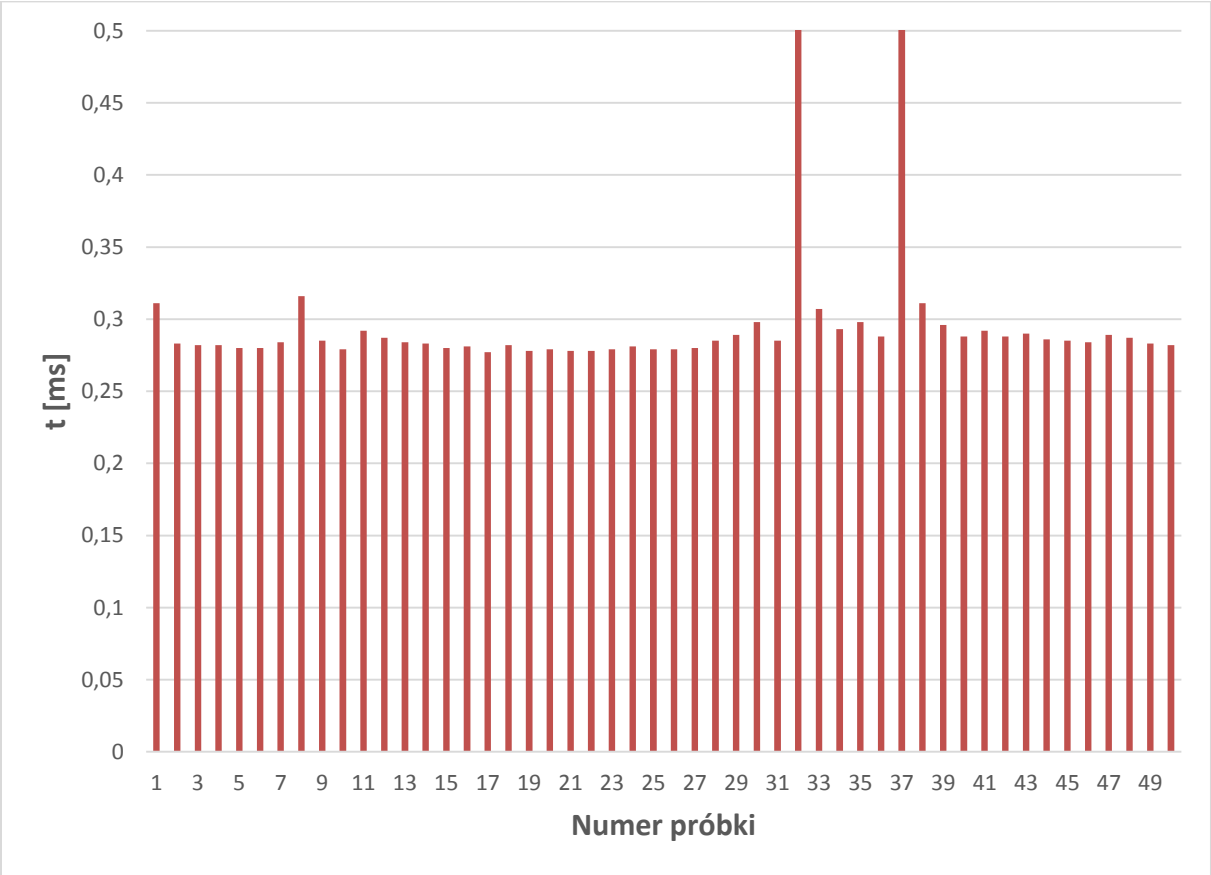
Wykres 2. T_z = 100 ms, $T_{on/off}$ = 50 ms

W przypadku gdy, częstotliwość PWM została obniżona 1000 krotnie do wartość 10 Hz, przy zachowaniu tych samych warunków pomiaru, to znaczy bez wątków obciążających procesor opóźnienia w stosunku do wartości zadanej były znacznie mniejsze (*Wykres 2*). Przy wartości zadanej zmiany wartości pinu 50 ms, otrzymana średnia wartość pomiaru wyniosła 50,5269 ms. Wynika z tego, że czas potrzebny na pozostałe obliczenia procesora związane z działaniem systemu wykonują się znacząco szybciej w stosunku do zmian wartości pinu i są praktycznie niezauważalne. Dodatkowo przy tym pomiarze uruchomione zostały wątki obciążające procesor, co odzwierciedlają skoki wychodzące poza skale wykresu, mające

wartość odpowiednio 51.57, 59.99, 52.54 51.63, 50,96. Dla czytelności wykresu zostały one pozostawione poza zakresem osi Y.

$$Delay = \left(\frac{50{,}5269 - 50}{50} * 100\%\right) = 1{,}05\ \%$$

- Częstotliwość = 2000 Hz; Okres = 0,5 ms; Wypełnienie 50%; Czas włączenia (5V) 0,25 ms; Czas wyłączenia (0V) 0,25 ms; Liczba próbek 50.



Wykres 3. T_z = 0,5 ms, $T_{on/off}$ = 0,25 ms

W dalszych pomiarach włączone zostały wątki obciążające z niskim priorytetem, zachowując przy tym wysoki priorytet wątku odpowiedzialnego za sterowanie pinem (*Wykres 3*). Częstotliwość została ustawiona na wartość 2000 Hz. Średnia wartość pomiaru wyniosła 0,4743 ms. Na tak dużą wartość złożyły się spore opóźnienia w pojedynczych próbkach, wynikające prawdopodobnie z dłuższej obsługi innych wątków, niż to powinno być. Widoczne na wykresie skoki osiągały wartość 1.94 ms oraz 8 ms.

$$Delay = \left(\frac{0{,}4743 - 0{,}25}{0{,}25} * 100\%\right) = 89{,}72\ \%$$

3.4.2 Pomiary oscyloskopem

Kolejnym etapem badań był pomiar przebiegu sygnały przy pomocy oscyloskopu. Pomiar ten odbywał się przy pomocy dwóch kanałów oscyloskopu. Jedna sonda podłączona była do pinu, na którym działał sprzętowy PWM (hard-PWM), zaś druga sonda podłączona była do pinu skonfigurowanego pod softwarowy PWM (soft-PWM). Badanie polegało na pomiarze przebiegu sygnału dla obu PWMów oraz porównaniu tych przebiegów. W trakcie badań mierzone były następujące parametry przebiegu sygnału:

- Częstotliwość
- Okres

Zaś badania odbyły się w dwóch konfiguracjach:

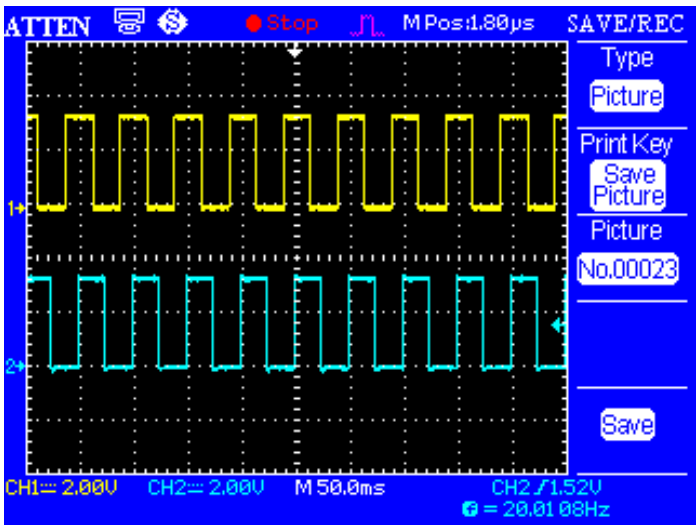
- Gdy wszystkie pozostałe wątki obciążające procesor były wyłączone
- Wraz z wątkami obciążającymi procesor

W pierwszej kolejności zbadany został przypadek z włączonymi wątkami obciążającymi procesor, w programie zdefiniowany był wątek uruchamiający PWM sprzętowy, wątek w którym działał PWM softwarowy oraz cztery pozostałe wątki mające na celu wykonywanie skomplikowanych dla procesora obliczeń, włączając w to dzielenie, które spośród wszystkich podstawowych operacji matematycznych jest najtrudniejsze dla procesora. Wyniki zamieszczone zostały w *Tabeli 1* (T_{zad} – Okres zadany, T_{uzy} – Okres pomierzony, D_{zad} – zdane wypełnienie).

Pomiar l.p.	T_{zad} [ms]	T_{uzy} [ms]		f_{zad} [Hz]	F_{uzy} [Hz]		D_{zad} [%]
		Soft-PWM	Hard-PWM		Soft-PWM	Hard-PWM	
1	100	100,2	100	10	9,98	10	50
2	100	100,2	100	10	9,98	10	10
3	50	50,2	50	20	19,92	20	50
4	50	50,2	50	20	19,92	20	10
5	25	25,16	25	40	39,75	40	50
6	25	25,16	25	40	39,75	40	10
7	10	10,18	10	100	98,23	100	50
8	10	10,18	10	100	98,23	100	10
9	5	5,18	5	200	193	200	50
10	5	5,17	5	200	193,4	200	10
11	2,5	2,67	2,48	400	374,2	403,2	50
12	2,5	2,67	2,48	400	374,2	403,2	10
13	1	1,16	0,984	1000	859,1	1020	50
14	1	1,17	0,984	1000	853,2	1020	10
15	0,5	0,664	0,48	2000	1510	2080	50
16	0,5	0,666	0,48	2000	1500	2080	10
17	0,25	0,423	0,24	4000	2360	4170	50
18	0,25	0,41	0,24	4000	2440	4170	10
19	0,1	0,262	0,093	10000	3810	10720	50

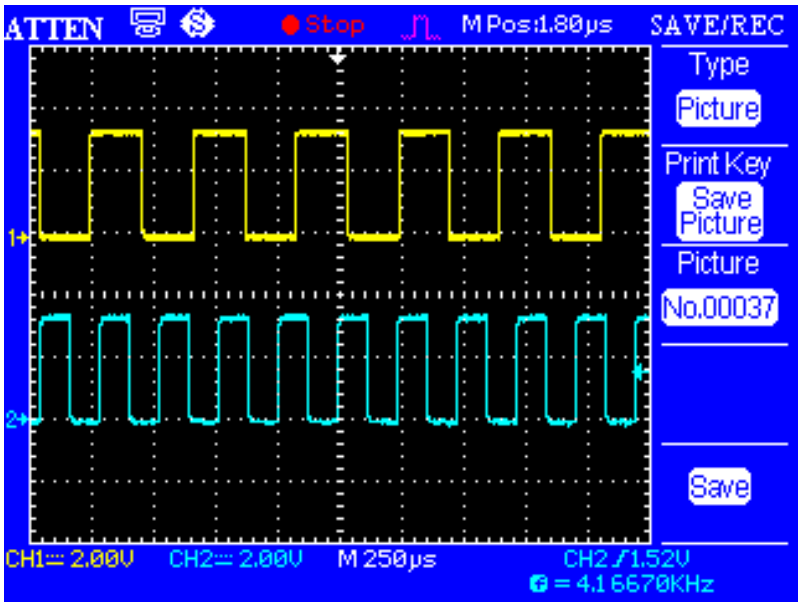
Tabela 1. Wyniki pomiaru okresu oraz częstotliwości z dodatkowym obciążeniem procesora.

Z powyższych pomiarów wynika, że sprzętowy PWM działa zdecydowanie lepiej od softwarowego. Programowy PWM praktycznie od samego początku pomiarów obarczony był błędem, podczas gdy sprzętowy do częstotliwości rzędu 200 Hz działał zgodnie z zadanymi parametrami. Początkowo oba PWMy działały podobnie i różnica była znikoma (*Rysunek 10*).



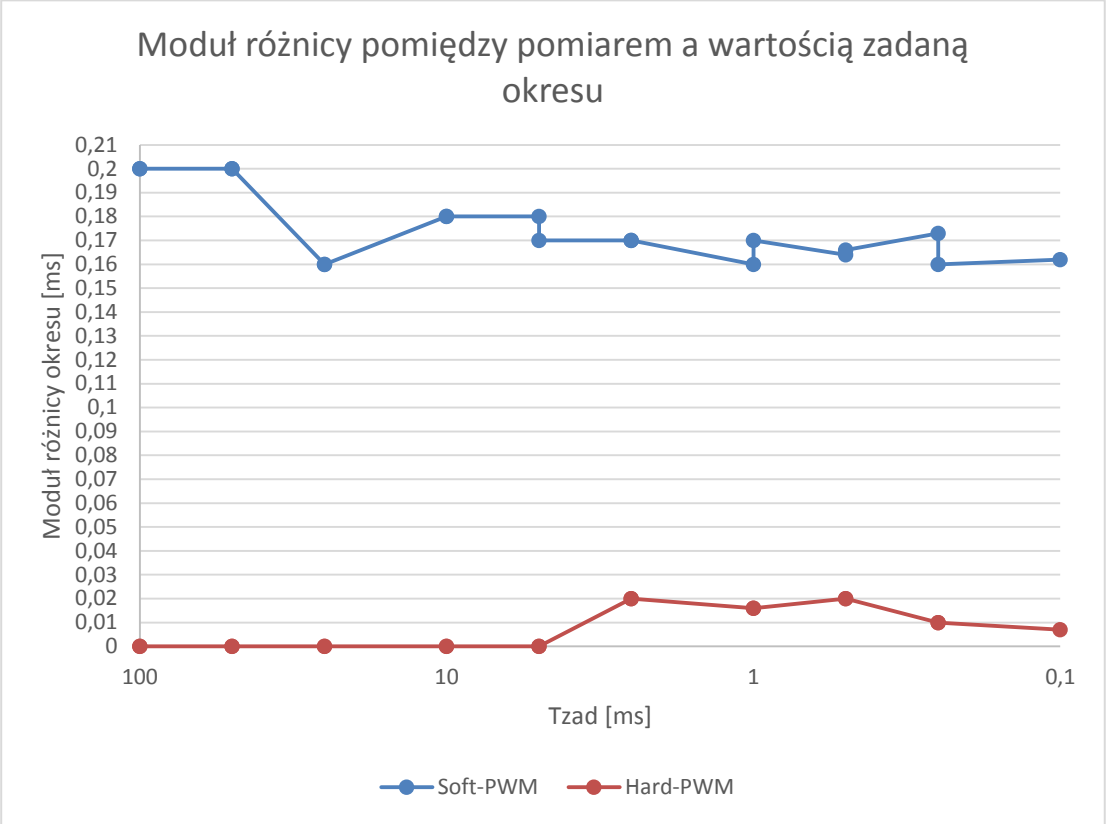
Rysunek 10. Pomiar przebiegu sygnałów PWM przy f=20 Hz

Natomiast pod koniec pomiaru przy wartościach częstotliwości znacznie wyższych niż początkowe różnica pomiędzy sprzętowym PWM a programowym była wyraźna, co widoczne jest na *Rysunku 11*, przy czym kolorem żółtym oznaczony jest przebieg sygnału programowego PWM, zaś niebieskim sprzętowego PWM.

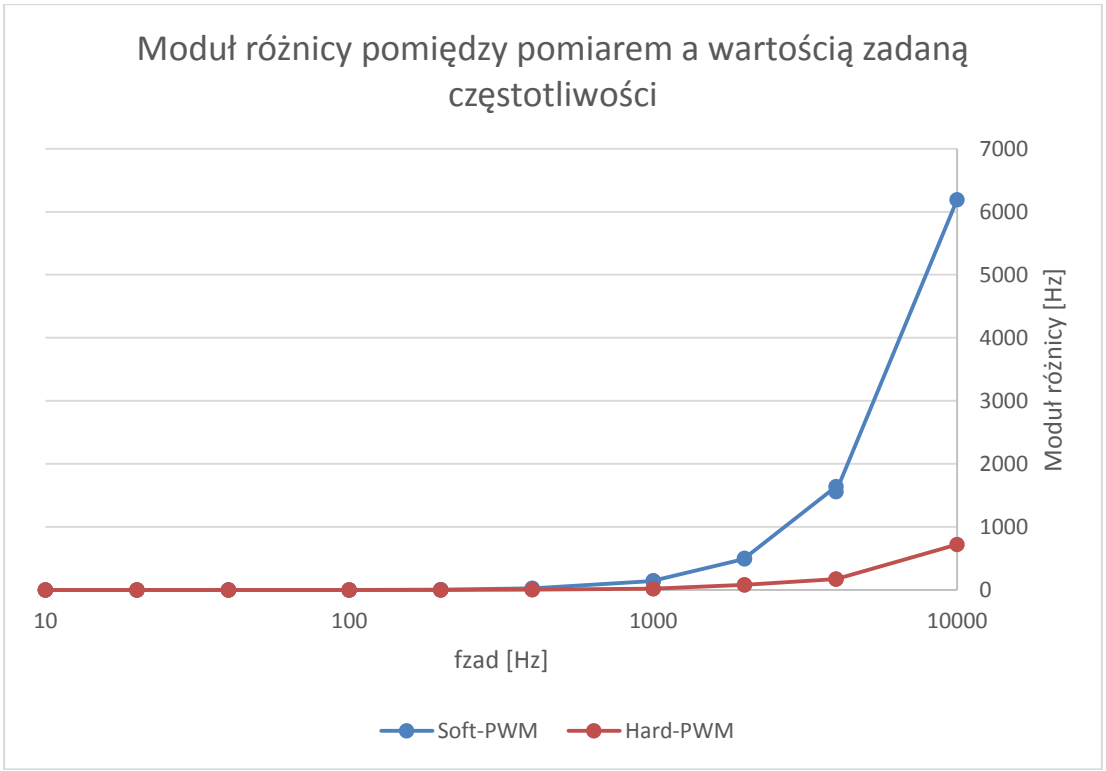


Rysunek 11. Pomiar przebiegu sygnałów PWM przy f=4 kHz

Dla otrzymanych pomiarów przeprowadzone zostały obliczenia różnicy pomiędzy wartościami zadanymi a zmierzonymi dla okresów i częstotliwości, dla obu PWMów, co zostało przedstawione na *Wykresach 1-2*.



Wykres 1. Moduł różnicy pomiędzy pomiarem a wartością zadaną okresu.



Wykres 2. Moduł różnicy pomiędzy pomiarem a wartością zadaną częstotliwości.

Następnie badanie zostało powtórzone, z tą różnicą, że tym razem wszystkie wątki mające na celu obciążanie procesora obliczeniami zostały wyłączone i w ten sposób zostały dwa: wątek uruchamiający sprzętowy PWM oraz wątek z programowym PWM. Wyniki pomiarów zamieszczone zostały w *Tabeli 2*.

Pomiar	**T_{zad} [ms]**	**T_{uzy} [ms]**		**f_{zad} [Hz]**	**F_{uzy} [Hz]**		**D_{zad} [%]**
l.p.		Soft-PWM	Hard-PWM		Soft-PWM	Hard-PWM	
1	100	100,2	100	10	9,98	10	50
2	100	100,2	100	10	9,98	10	10
3	50	50,2	50	20	19,92	20	50
4	50	50,2	50	20	19,92	20	10
5	25	25,2	25	40	39,68	40	50
6	25	25,1	25	40	39,84	40	10
7	10	10,18	10	100	98,23	100	50
8	10	10,18	10	100	98,23	100	10
9	5	5,18	5	200	193	200	50
10	5	5,19	5	200	192,6	200	10
11	2,5	2,66	2,49	400	375,9	401,2	50
12	2,5	2,67	2,49	400	374,5	401,2	10
13	1	1,16	1	1000	862	1000	50
14	1	1,17	1	1000	856,1	1000	10
15	0,5	0,67	0,493	2000	1490	2030	50
16	0,5	0,666	0,493	2000	1500	2030	10
17	0,25	0,414	0,24	4000	2420	4170	50
18	0,25	0,416	0,24	4000	2400	4170	10
19	0,1	0,27	0,093	10000	3700	10730	50

Tabela 2. Wyniki pomiaru okresu oraz częstotliwości, bez obciążenia procesora.

Z powyższych pomiarów wynika, że wyłączenie wątków obciążających procesor nie wpłynęło znacząco na poprawę wyników pomiarów dla programowego PWM. Otrzymane rezultaty są niekiedy nieco gorsze niż w poprzednim badaniu, aczkolwiek tak małe różnice można uznać za pomijalnie małe, zatem wyniki pomiaru były bardzo zbliżone.

Z badań przeprowadzonych dla systemu, który nie spełnia założeń systemu czasu rzeczywistego, wynika że używając standardowego jądra oraz POSIXowych wątków systemu nie jesteśmy w stanie uzyskać przebiegów sygnału zgodnych z wartościami zadanymi. System nie nadąża z obsługą programu, który miał za zadanie zmianę wartości napięcia na pinie. Z

badań wynika, że PWM, działający niezależnie od procesora jest w stanie osiągnąć znacznie lepsze rezultaty niż PWM programowy uruchomiony na takim systemie.

Rozdział 4.

Konfiguracja systemu

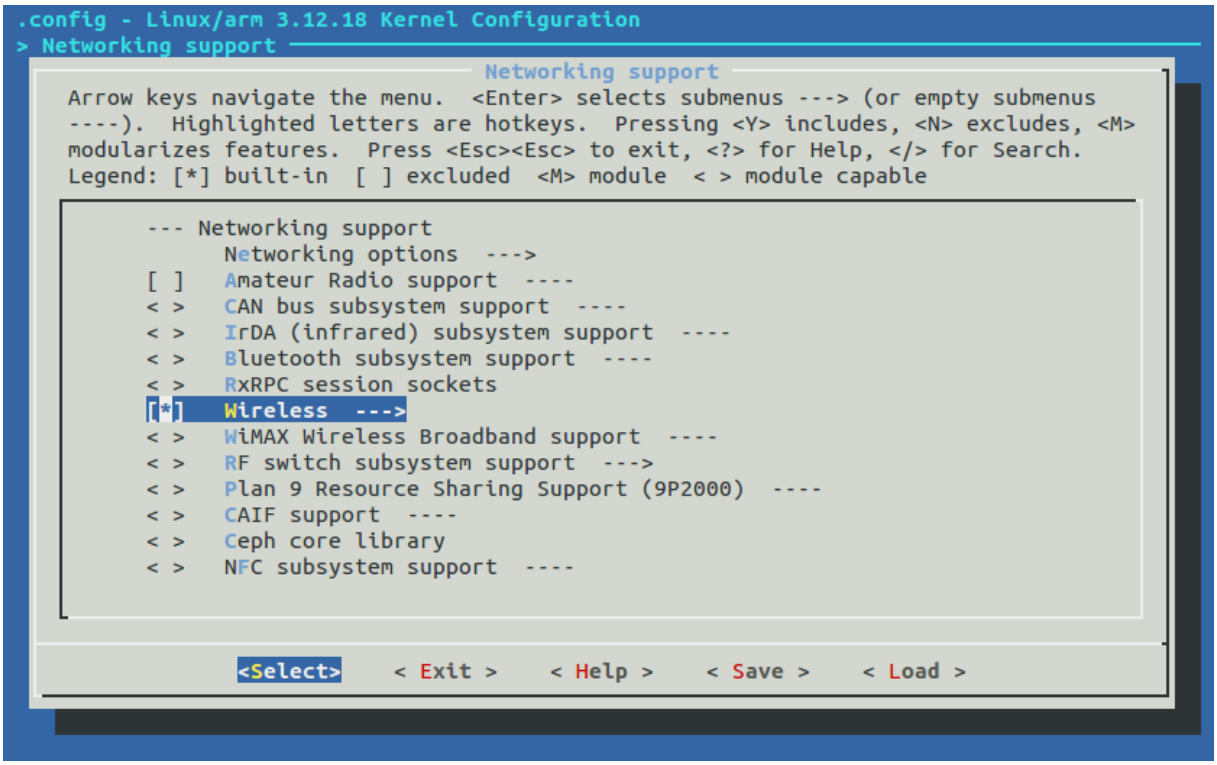
W celu uzyskania przebiegu sygnału PWM programowego bardziej zbliżonego do wartości zadanych skonfigurowany został system czasu rzeczywistego na bazie Xenomai z jądrem Linux. Ze względu na ograniczoną dostępność Pipe [5] patchów dla Xenomai oraz Linux, użyta została stosunkowo stara wersja jądra Linux jako bazy do zbudowania systemu. Nie udało się znaleźć odpowiedniej nakładki dla pipe'ów nowszych wersji jądra, które współdziałałyby z Xenomai. Z tego względu do budowy systemu posłużyło jądro Linux z serii 3.8.y i wersja Xenomai z serii 2.6.

4.1 Środowisko

W pierwszej kolejności za środowisko do budowy nowego systemu posłużyło Raspberry PI w działającym ówcześnie systemem Raspbian. W tym przypadku kompilacja systemu czasu rzeczywistego odbywała się bezpośrednio w środowisku docelowym. Po wielu nieudanych próbach, podjęta została decyzja o zmianie środowiska kompilacji na zewnętrzne. Powodem takiej decyzji była między innymi niska wydajność sprzętowa Raspberry PI w porównaniu do standardowych sprzętów klasy PC. Czas każdorazowej kompilacji systemu wynosił od 6 do 9 godzin, zaś po zmianie środowiska na maszynę Linuxową dystrybucji Ubuntu, zainstalowaną na laptopie o znacznie wydajniejszym procesorze czas kompilacji skrócił się do około 2-3 godzin. Budowa nowego systemu w środowisku innym niż docelowe wymaga tak zwanej kompilacji skrośnej (Cross-Compilation), wymaga to pobrania specjalnego kompilatora dedykowanego do środowiska docelowego.

[5] Potok (pipe) – mechanizm do komunikacji między procesorowej

W ramach prób budowy systemu wykorzystany został Buildroot, który jest dedykowanym narzędziem do budowy systemów z użyciem kompilacji skrośnej. Narzędzie to po odpowiednim skonfigurowaniu pobiera potrzebne moduły i patche, niezbędne do budowy systemu, następnie przygotowuje konfigurację systemu oraz wykonuje jego budowę (*Rysunek 12*).



Rysunek 12. Przykładowe okno narzędzia Buildroot [20]

Niestety nie udało się zbudować poprawnie systemu przy pomocy tego narzędzia. Ponieważ Buildroot dostarcza wiele możliwości modyfikowania i konfigurowania systemu docelowego, niektóre z ustawień jakie zostały wybrane mogły nie współdziałać z innymi, zaś najczęstszą przyczyną przerwanej kompilacji systemu była niemożliwość znalezienia przez narzędzie odpowiedniej wersji pipe patcha. Narzędzie posiada także pewne ograniczenie w postaci kompatybilności z określonym zakresem wersji jąder Linuxa oraz Xenomai, więc niestety gdy dla danego połączenia wersji jądra z Xenomai znajdował się właściwy pipe patch, to narzędzie nie obsługiwało tych wersji systemu. Natomiast gdy użyta została inna wersja

Buildroota, posiadająca inny zakres kompatybilności z wersjami jądra, pojawiały się inne problemy z kompilacją. Z tego też względu podjęta została decyzja o budowie systemu bez dodatkowych narzędzi, poprzez własnoręczne wpisywanie komend w konsoli.

4.2 Budowa systemu

Budowa systemu jest skomplikowanym procesem wymagającym wielu przygotowań oraz czasu poświęconego na jego przygotowanie oraz kompilację. Możliwości i sposobów na zbudowanie systemu na bazie Xenomai jest wiele, nie mniej jednak, każdy z nich dzieli się na następujące etapy:

1. Pobranie niezbędnych pakietów oraz plików źródłowych
2. Instalacja nakładek (pipe patch) do jądra
3. Konfiguracja jądra
4. Kompilacja systemu
5. Instalacja systemu na środowisku docelowym

Na etap pierwszy w rozpatrywanym przypadku budowy systemu na bazie Xenomai z jądrem Linuxa składa się pobranie pakietów Xenomai, jądra Linuxa, kompilatora skrośnego oraz pipe patchów. Dla potrzeb przyszłych prac nad tym systemem przytoczono dokładnie polecenia niezbędne do realizacji poszczególnych etapów.

Pobranie kompilatora

```
wget https://github.com/raspberrypi/tools/archive/master.tar.gz
```

Pobranie jądra Linuxa dla Raspberry PI

```
git clone -b rpi-3.8.y --depth 1 git://github.com/raspberrypi/linux.git linux-rpi-3.8.y
```

Pobranie pakietu Xenomai

```
git clone git://git.xenomai.org/xenomai-head.git Xenomai-head
```

Pobranie Pipe Patchów – dostępne na stronie *http://xenomai.org/downloads/ipipe/*

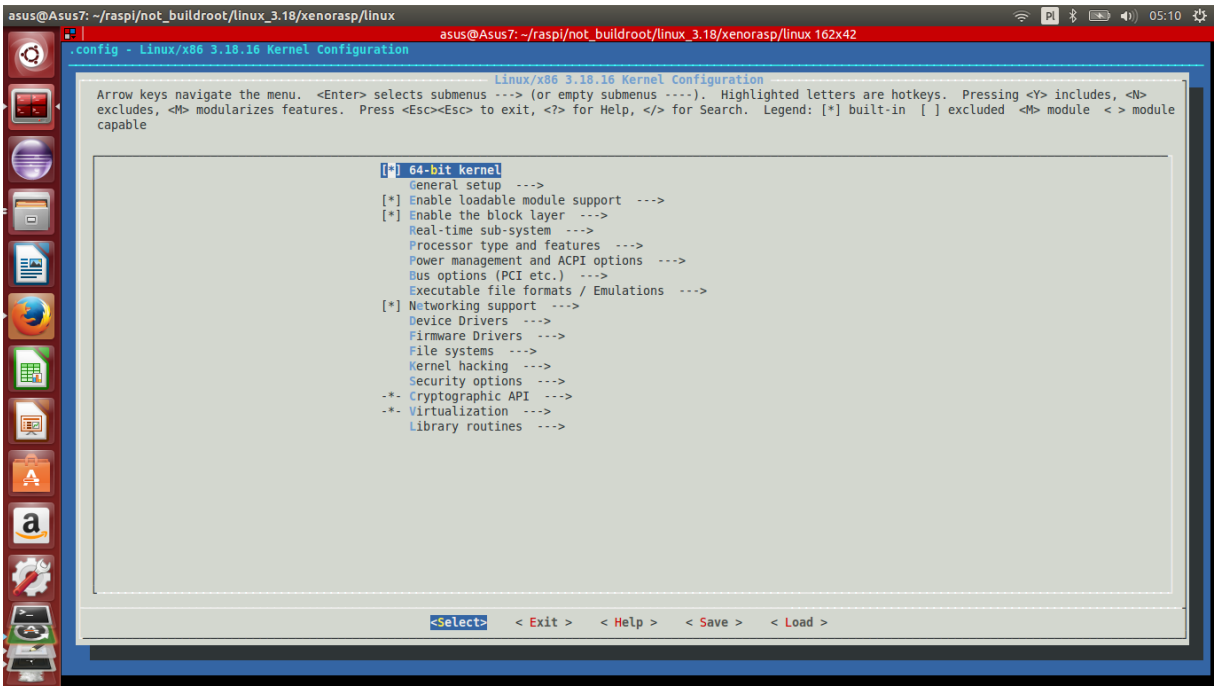
Etap drugi polega na zaaplikowaniu pakietu Xenomai do plików źródłowych jądra oraz dwóch nakładek – pre-patch oraz post-patch w odpowiedniej kolejności.

```
cd linux-rpi-3.8.y

patch -Np1 < ../xenomai-head/ksrc/arch/arm/patches/raspberry/ipipe-core-3.8.13-raspberry-pre-2.patch

xenomai-head/scripts/prepare-kernel.sh --arch=arm --linux=linux-rpi-3.8.y --adeos=xenomai-head/ksrc/arch/arm/patches/ipipe-core-3.8.13-arm-3.patch

patch -Np1 < ../xenomai-head/ksrc/arch/arm/patches/raspberry/ipipe-core-3.8.13-raspberry-post-2.patch
```

Gdy jądro Linuxowe zostanie przygotowane poprzez zainstalowanie nakładek pozwalających na komunikację pomiędzy jądrem Linuxa a jądrem Xenomai, następuję etap konfiguracji systemu, polegający na nadaniu mu odpowiednich parametrów, wyborze modułów, architektury docelowej, dostępnych bibliotek systemowych oraz wielu innych parametrów. W sieci można pobrać gotowe konfiguracje, które można w prosty sposób użyć, jednakże możliwe jest też samodzielne skonfigurowanie systemu na bazie podstawowej konfiguracji. Odbywa się to poprzez wywołanie następującej komendy:

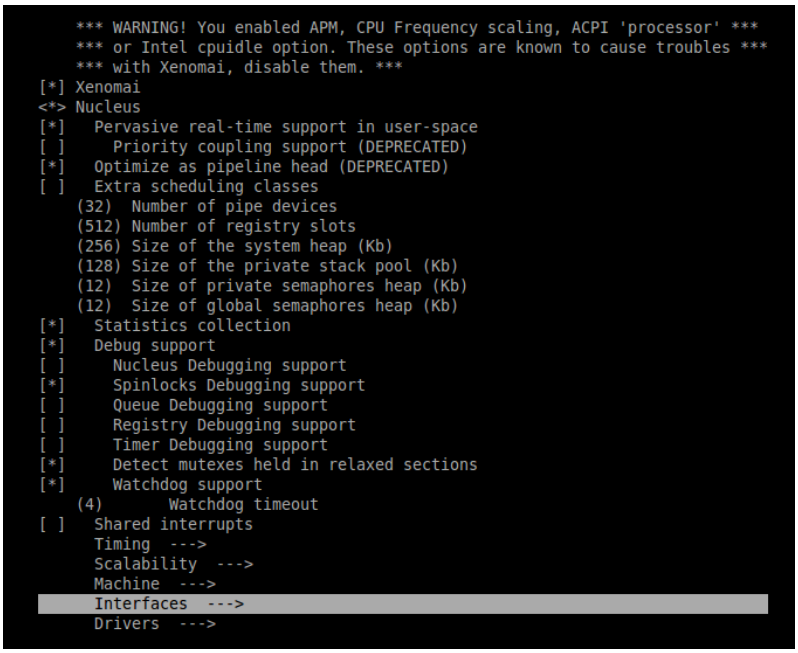
```
make mrproper

make menuconfig
```

Następnie ukazuje się okno, w którym można przygotować nową konfigurację (*Rysunek 13*) bazującą na minimalnych ustawieniach, niezbędnych do działania systemu.

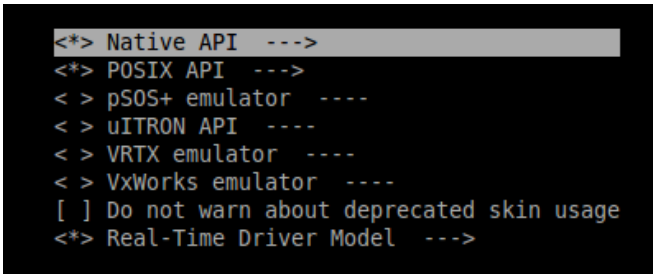


Rysunek 13. Okno konfiguracji systemu

W konfiguracji należy podać architekturę systemu docelowego, w tym przypadku jest to architektura ARM. Następnie w podmenu „Real-time sub-system” należy zaznaczyć Xenomai, Nucleus oraz Pervasive real-time suport in user-space (*Rysunek 14*). Następnie należy przejść do podmenu Interfaces i wybrać w nim Native API oraz opcjonalnie POSIX API (*Rysunek 15*). Jeżeli w etapie drugim poprawnie zostały zaaplikowane nakładki, minimalna konfiguracja powinna być przygotowana automatycznie, należy się tylko upewnić, że podstawowe parametry są właściwe.



Rysunek 14. Podmenu Real-time sub-system



Rysunek 15. Podmenu Real-time sub-system -> Interface

Jest to niezbędne do działania systemu z jądrem Xenomai. Podana konfiguracja jest minimalną, możliwości dostosowywania systemu do własnych potrzeb jest bardzo wiele i nie sposób jest zawrzeć je wszystkie w pracy. W trakcie przygotowywania systemu przetestowanych zostało wiele możliwych konfiguracji i zdecydowana większość z nich generowała błędy w trakcie kompilacji systemu. Ostatecznie po wielu próbach budowania systemu sprowadziło się to do konfiguracji minimalnej, aby system był w stanie się skompilować, a większość zbędnych modułów została wyłączona. Niestety jak się później okazało wyłączone zostało także wsparcie dla układów peryferyjnych procesora Raspberry PI w tym także sprzętowego PWM. Nie mniej jednak nie miało to wpływu na badania, gdyż ten

rodzaj PWMa działa niezależnie od systemu (i został zbadany wcześniej) wiec jego zmiana nie powinna mieć wpływu na działanie PWM.

Gdy już konfiguracja zostanie zakończona należy zamknąć okno i zapisać konfigurację. Następnym etapem budowania systemu jest jego kompilacja. Aby wykonać prawidłową kompilację należy użyć kompilatora skrośnego, który został pobrany w pierwszym etapie. Do wyboru są cztery kompilatory, których można użyć do tego celu (*Rysunek 16*). Większość instrukcji jakie można znaleźć w sieci preferuje kompilatory arm-bcm2708, które przez długi czas były wykorzystywane w tym projekcie do budowy systemu. Kompilator ten występuje w dwóch wersjach hard floating point[6] oraz soft floating point[7] suport. Jak się z czasem okazało zdecydowana większość problemów z kompilacją systemu miała związek właśnie z soft/hard floating point. Pomimo, iż w konfiguracji zaznaczany był parametr hard floating point, po użyciu kompilatora arm-bcm2708hardfp pojawiał się błąd, że nie można użyć kompilatora z hard floating point do soft floating point. Po wielu próbach różnych kombinacji użyć tego modułu zaniechane zostały próby użycia tego kompilatora.

```
[06:24][asus@Asus7][~/raspi/not_buildroot/linux_3.8/tools-master/arm-bcm2708] $ ll
total 24
drwxrwxr-x 6 asus asus 4096 cze  2  2015 ./
drwxrwxr-x 8 asus asus 4096 cze  2  2015 ../
drwxrwxr-x 7 asus asus 4096 cze  2  2015 arm-bcm2708hardfp-linux-gnueabi/
drwxrwxr-x 7 asus asus 4096 cze  2  2015 arm-bcm2708-linux-gnueabi/
drwxrwxr-x 7 asus asus 4096 cze  2  2015 gcc-linaro-arm-linux-gnueabihf-raspbian/
drwxrwxr-x 7 asus asus 4096 cze  2  2015 gcc-linaro-arm-linux-gnueabihf-raspbian-x64/
[06:24][asus@Asus7][~/raspi/not_buildroot/linux_3.8/tools-master/arm-bcm2708] $
```

Rysunek 16. Dostępne skrośne kompilatory dla Raspberry PI

W dalszych działaniach użyty został kompilator linaro (gcc-linaro-arm-linux-gnueabihf-raspbian-x64), który jak się później okazało były jedynym działającym poprawnie

[6] Hard floating point – używa dedykowanego modułu w procesorze do obliczeń zmiennoprzecinkowych (jest znacznie szybsze, ale nie każdy procesor posiada taki moduł)
[7] Soft floating point emuluje programowo obliczenia zmiennoprzecinkowe

dla podanej konfiguracji. Po wybraniu odpowiedniego kompilatora należy wykonać komendę rozpoczynającą kompilację nowego systemu z odpowiednimi parametrami.

```
make ARCH=arm O=build CROSS_COMPILE=/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-
```

Po wykonaniu komendy nastąpi rozpoczęcie budowy systemu, które może potrwać do kilku godzin. Gdy kompilacja zostanie ukończona należy jeszcze zainstalować moduły nowego jądra w bibliotece systemu, w której znajdują się wszystkie dodatki i modyfikacje do standardowego jądra systemu.

```
make ARCH=arm O=build INSTALL_MOD_PATH=dist modules_install
```

Następnie należy doinstalować pliki nagłówkowe do modułów jądra, poprzez wykonanie komendy:

```
make ARCH=arm O=build INSTALL_HDR_PATH=dist headers_install
```

Efektem końcowym budowy systemu jest powstanie pliku z obrazem systemu, który następnie należy zainstalować w docelowym środowisku. Plik „zImage” powinien znajdować w następującej lokalizacji:

```
../linux-rpi-3.8.y/build/arch/arm/boot/
```

Plik ten należy umieścić na karcie SD służącej jako dysk Raspberry PI w partycji „boot”.

Gdy zostanie to już wykonane, na urządzeniu docelowym, którym jest Raspberry PI należy jeszcze pobrać pakiet Xenomai i zainstalować go bezpośrednio na urządzeniu, aby mieć dostęp do pełnych zasobów nakładki Xenomai.

```
wget http://download.gna.org/xenomai/stable/xenomai-2.6.4.tar.bz2
tar -jxvf xenomai-2.6.4.tar.bz2
cd xenomai-2.6.4
apt-get update build-essentials
./configure
make
sudo make install
```

Jeżeli instalacja przebiegła poprawnie, system powinien być poprawnie skonfigurowany z pełnym dostępem do zasobów. Aby upewnić się, że system został zbudowany poprawnie i wszystkie moduły działają prawidłowo, należy po zalogowaniu się do urządzenia poprzez ssh, sprawdzić wersję jądra systemu oraz dodatków w postaci Xenomai (*Rysunek 17 -18*).

```
pi@raspberrypi / $ cat /proc/version
Linux version 3.8.13-core+ (asus@Asus7) (gcc version 4.8.3 20140303 (prerelease) (crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC 2014.03) ) #2 Fri Jan 29 00:22:3
8 CET 2016
pi@raspberrypi / $ cat /proc/xenomai/version
2.6.4
```

Rysunek 17. Sprawdzenie wersji jądra systemu oraz dodatków (Xenomai)

```
pi@raspberrypi / $ dmesg | grep Linux
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 3.8.13-core+ (asus@Asus7) (gcc version 4.8.3 20140303 (prerelease) (crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC 2014.03) ) #2 Fri
 Jan 29 00:22:38 CET 2016
[    1.770556] usb usb1: Manufacturer: Linux 3.8.13-core+ dwc_otg_hcd
pi@raspberrypi / $ dmesg | grep Xenomai
[    1.105449] I-pipe: head domain Xenomai registered.
[    1.110390] Xenomai: hal/arm started.
[    1.114260] Xenomai: scheduling class idle registered.
[    1.119414] Xenomai: scheduling class rt registered.
[    1.129102] Xenomai: real-time nucleus v2.6.4 (Jumpin' Out) loaded.
[    1.135465] Xenomai: debug mode enabled.
[    1.139839] Xenomai: starting native API services.
[    1.144772] Xenomai: starting RTDM services.
```

Rysunek 18. Sprawdzenie procesów systemowych związanych z jądrem Linux oraz Xenomai

Tak przygotowany system jest gotowy do działania i możliwe jest korzystanie w pełni z funkcjonalności Xenomai-API.

Rozdział 5.

Testy systemu

Po zbudowaniu oraz poprawnym zainstalowaniu systemu z jądrem Xenomai, możliwe było przeprowadzenie badań oraz pomiarów osiągów systemu, w celu zweryfikowania jego działania w porównaniu do standardowego systemu opisanego w Rozdziale 3. Przebieg badań był zbliżony do opisanego we wspomnianym rozdziale. W ramach badań powstała aplikacja na bazie stworzonej wcześniej, która miała na celu wykonywanie tej samej czynności jaką było generowanie PWM, z tą różnicą że odbywało się to przy pomocy modułów Xenomai, a zatem w reżimie czasu rzeczywistego.

5.1 Pomiary opóźnienia w systemie

W pierwszej kolejności do badań wykorzystany został skrypt dostarczany wraz z pakietem Xenomai przez producenta. W przestrzeni użytkownika znajduje się aplikacja mierząca opóźnienie systemu, polega ona na utworzeniu zadania czasu rzeczywistego w przestrzeni użytkownika. Działanie polega na periodycznym wywoływaniu tego zadania co 1000 mikrosekund i pomiarze opóźnienia. Priorytet tego zadania ustawiony jest na najwyższy. Wyniki pomiaru zostały pokazane na *Rysunku 19*.

```
pi@raspberrypi /usr/xenomai/bin $ sudo ./xeno latency
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      5.064|      8.148|     36.416|       0|     0|      5.064|     36.416
RTD|      4.436|      8.184|     38.060|       0|     0|      4.436|     38.060
RTD|      4.520|      8.444|     39.852|       0|     0|      4.436|     39.852
RTD|      5.548|      8.536|     40.012|       0|     0|      4.436|     40.012
RTD|      5.348|      8.012|     38.640|       0|     0|      4.436|     40.012
RTD|      5.360|      8.060|     37.080|       0|     0|      4.436|     40.012
RTD|      5.360|      8.196|     39.380|       0|     0|      4.436|     40.012
RTD|      5.368|      8.044|     35.804|       0|     0|      4.436|     40.012
RTD|      5.396|      8.304|     37.136|       0|     0|      4.436|     40.012
RTD|      5.236|      8.440|     40.756|       0|     0|      4.436|     40.756
RTD|      5.380|      8.596|     34.404|       0|     0|      4.436|     40.756
RTD|      5.140|      8.184|     36.340|       0|     0|      4.436|     40.756
RTD|      5.436|      8.072|     37.944|       0|     0|      4.436|     40.756
RTD|      5.080|      7.968|     37.384|       0|     0|      4.436|     40.756
RTD|      5.268|      8.148|     40.340|       0|     0|      4.436|     40.756
RTD|      5.408|      8.228|     38.740|       0|     0|      4.436|     40.756
RTD|      5.340|      8.248|     36.824|       0|     0|      4.436|     40.756
RTD|      5.156|      8.616|     37.676|       0|     0|      4.436|     40.756
RTD|      5.136|      8.584|     36.872|       0|     0|      4.436|     40.756
RTD|      5.404|      8.044|     38.340|       0|     0|      4.436|     40.756
RTD|      5.148|      8.200|     37.360|       0|     0|      4.436|     40.756
RTT|  00:00:22  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      5.244|      8.140|     40.696|       0|     0|      4.436|     40.756
RTD|      5.336|      8.216|     36.876|       0|     0|      4.436|     40.756
RTD|      5.216|      8.496|     36.100|       0|     0|      4.436|     40.756
RTD|      5.024|      8.736|     39.688|       0|     0|      4.436|     40.756
RTD|      4.752|      8.820|     40.020|       0|     0|      4.436|     40.756
RTD|      5.376|      8.468|     35.960|       0|     0|      4.436|     40.756
RTD|      4.944|      8.336|     38.244|       0|     0|      4.436|     40.756
^C---|-----------|-----------|-----------|--------|------|-------------------------
RTS|      4.436|      8.300|     40.756|       0|     0|    00:00:29/00:00:29
pi@raspberrypi /usr/xenomai/bin $
```

Rysunek 19. Pomiar opóźnienia zadania czasu rzeczywistego.

Jak widać na powyższym rysunku przy zadaniu wywoływanym co 1000 µs, opóźnienia zawierały się w granicach 4,436 – 40,756 us, uzyskując średnią wartość 8,3 us.

$$Delay = \left(\frac{8{,}3}{1000} * 100\%\right) = 0{,}83\ \%$$

Uzyskany wynik dla okresu wynoszącego 1 ms, jest lepszy od wyniku jaki udało się uzyskać dla standardowego systemu (non-RT) gdzie przy okresie wynoszącym 100 ms wartości opóźnienia były zbliżone do 1%. Należy jednak zwrócić uwagę, że w przypadku obecnego testu, pomiar odbywał się bezpośrednio w systemie, zaś w przypadku standardowego systemu pomiar wykonywany był przez aplikację testującą.

5.2 Aplikacja testująca

Aplikacja do celów pomiarowych została stworzona na bazie programu, który był wykorzystany w systemie standardowym. W nowej aplikacji stworzone zostały wątki do obsługi PWM oraz wątki mające na celu obciążenie procesora, w aplikacji zrezygnowano z modułu zbierającego stemple czasowe, gdyż poprzednie badania pokazały, że sam fakt zapisu stempli czasowych mógł mieć wpływ na uzyskiwane wartości, wiec niejako był zakłóceniem dla systemu mającego na celu generowanie PWMa. Główna zmiana w programie polegała na stworzeniu dodatkowego wątku czasu rzeczywistego tworzonego przy pomocy Xenomai-API. Wątki obciążające pozostały tworzone przy pomocy metod POSIXowych z niskim priorytetem działania. Również pozostawiony został wątek POSIXowy odpowiedzialny z PWM w celach porównawczych. Z uruchamiania wątku Hard-PWM zrezygnowano, gdyż działa on niezależnie od systemu, wiec wyniki jego działania pozostaną bez zmian. Aplikacja uruchamiała moduł inicjalizacyjny, w którym następowało tworzenie wątku czasu rzeczywistego oraz jego uruchomienie.

```
if(init_module()) {
        std::cout << "Init done" << std::endl;
}
```

```
int init_module(void) {
        int err1;
        err1 = rt_task_create(&task_desc2, "SoftPWM", TASK_STKSZ, TASK_PRIO,
        TASK_MODE);
        if(!err1) {
                rt_task_start(&task_desc2, &gpioThread, NULL);
        }
        return 1;
}
```

Użyte w tym celu funkcje tworzące i startujące wątki, są częścią interfejsu Native Xenomai API – Task management services. Samo działanie programu wewnątrz wątków

pozostało praktycznie bez zmian, z tą różnicą, że w przypadku PWM programowego pętla zajmująca się sterowaniem wartością napięcia na pinie działała nieskończenie długo, to znaczy dopóki wątek, w którym pętla ta była wykonywana, nie został przerwany. Dodatkowo zmieniony został sposób odmierzania czasu pomiędzy zmianami wartości pinu, tj zamiast usleep() został użyty rt_task_sleep() zaczerpnięty z Xenomai-API. Wątek ten, przerwać można było naciskając dowolny klawisz po uruchomieniu aplikacji. Ponieważ aplikacja ta została napisana w przestrzeni użytkownika, wszystkie jej wątki także działały w tej samej przestrzeni. Ponieważ tym razem tworzone zostały dwa wątki do sterowania PWM, na różne sposoby, potrzebne było stworzenie dwóch osobnych metod do ich uruchamiania. Wątek PWM real-time działał na pinie 7, zaś wątek POSIX PWM działał na pinie 8. Sposób działania wątku real-time z użyciem Xenomai-API:

```
void gpioC::gpioPWMStart() {
        double period = 1 / freq;
        double timeOn  = (( period * duty ) / 100) * 1000000;
        double timeOff = (( period * (100 - duty)) / 100) * 1000000;

        cout << "Debug timeon " << timeOn << " us"<< endl;
        gpioExportPin(this->pin);

        double timeOnX = timeOn * 1000 * 4; //skalowanie
        double timeOddX = timeOff * 1000 * 4; //skalowanie

        RTIME timeOnX_tic = (RTIME)rt_timer_ns2tsc(timeOnX);
        RTIME timeOffX_tic = (RTIME)rt_timer_ns2tsc(timeOffX);

        while(1) {
                gpioSet();
                rt_task_sleep(timeOnX_tic);
                gpioClear();
                rt_task_sleep(timeOffX_tic);
        }
        gpioUnExportPin(this->pin);
}
```

Sposób działania wątku POSIX:

```
void gpioC::gpioPWMStart2() {
        double period = 1 / freq;
        double timeOn  = (( period * duty ) / 100) * 1000000;
        double timeOff = (( period * (100 - duty)) / 100) * 1000000;

        cout << "Debug timeon " << timeOn << " us"<< endl;
        gpioExportPin(this->pin);

        while(1) {
                gpioSet();
                usleep(timeOn);
                gpioClear();
                usleep(timeOff);
        }
        gpioUnExportPin(this->pin);
}
```

W nowych wątkach brakuje elementu używanego wcześniej to znaczy czekania programowo, aż pin osiągnie zadaną wartość:

```
while(!gpioReadPin(this->pin));
```

Zostało to celowo usunięte, gdyż w trakcie badań okazało się, że metoda ta wprowadza znaczące opóźnienie w działaniu aplikacji, a jednocześnie ze względu na pomiar oscyloskopem nie jest potrzebne sprawdzanie stanu pinu programowo.

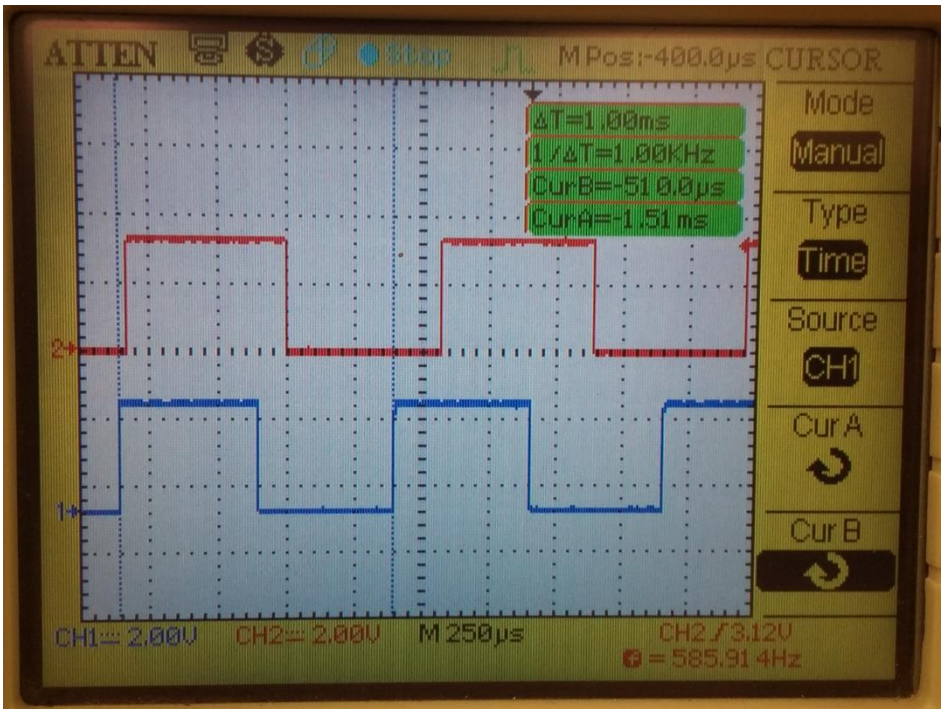
5.3 Pomiary

Pomiary przebiegu sygnału dla programowego PWM odbyły się tak jak w przypadku wcześniejszych badań przy użyciu oscyloskopu, dla tych samych wartości częstotliwości zadanych oraz okresów zadanych. Badanie zostało przeprowadzone dla programowego PWM z użyciem wątku czasu rzeczywistego oraz wątku POSIXowego, a wyniki tego pomiaru zostały zamieszczone w *Tabeli 3*. Wątki obciążające procesor zostały wyłączone na czas pomiaru.

Pomiar l.p.	T_{zad} [ms]	T_{uzy} [ms]		f_{zad} [Hz]	F_{uzy} [Hz]		D_{zad} [%]
		Xenomai	POSIX		Xenomai	POSIX	
1	100	100	100	10	10	10	50
2	100	100	100	10	10	10	10
3	50	50	50	20	20	20	50
4	50	50	50	20	20	20	10
5	25	25	25,2	40	40	39,68	50
6	25	25	25,2	40	40	39,68	10
7	10	10,1	10,2	100	99,01	98,04	50
8	10	10,1	10,2	100	99,01	98,04	10
9	5	5,04	5,08	200	198,4	196,8	50
10	5	5,04	5,08	200	198,4	196,8	10
11	2,5	2,52	2,54	400	396,8	393,7	50
12	2,5	2,51	2,55	400	398,4	392,16	10
13	1	1	1,06	1000	1000	943,4	50
14	1	1,01	1,07	1000	990,1	934,5	10
15	0,5	0,504	0,556	2000	1984,13	1798,56	50
16	0,5	0,504	0,562	2000	1984,13	1779,36	10
17	0,25	0,26	0,31	4000	3846,15	3225,81	50
18	0,25	0,254	0,323	4000	3937,01	3095,97	10
19	0,1	0,107	0,218	10000	9345,79	4587,16	50
20	0.1	0,107	0,168	10000	9345,79	5952,38	10
21	0,05	0,049	0,226	20000	20408,16	4424,78	50
22	0,05	0,053	0,23	20000	18867,92	4347,83	10

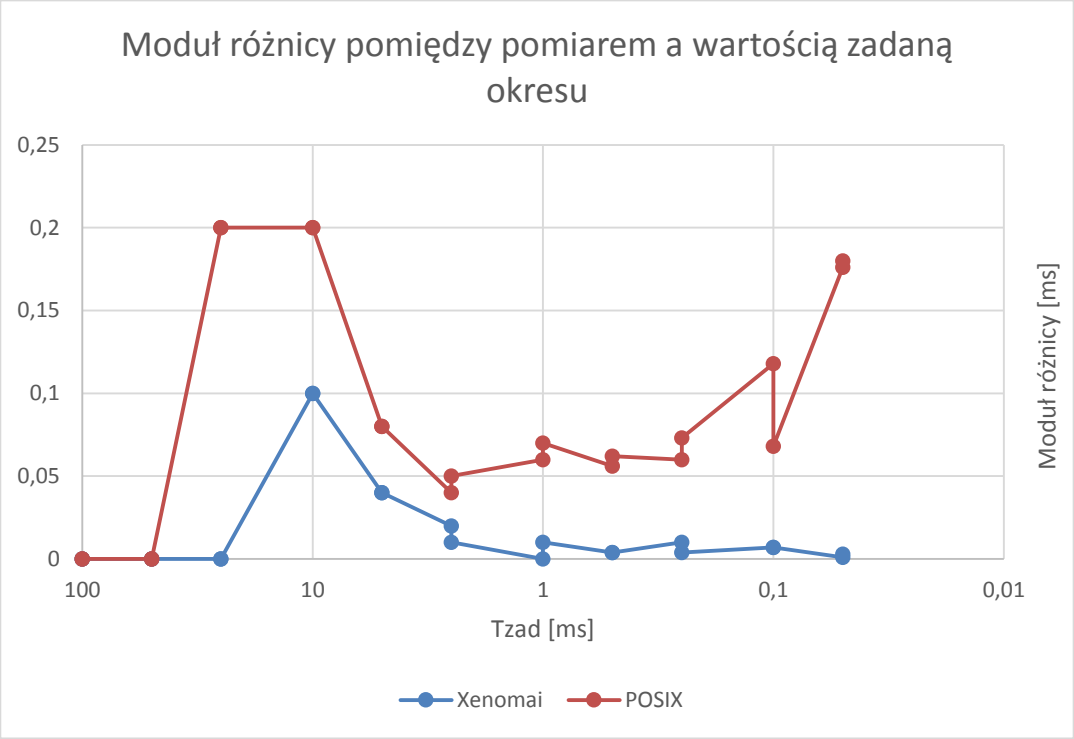
Tabela 3. Wyniki pomiaru okresu oraz częstotliwości PWM dla jądra Xenomai.

Wyniki pomiarów pokazały, że brak programowego sprawdzania stanu pinu znacznie poprawił osiągane rezultaty zarówno dla PWM z użyciem Xenomai jak i POSIX. Niemniej jednak na podstawie uzyskanych pomiarów wyraźnie widać, że wątek stworzony z użyciem Xenomai-API działa znacznie lepiej od wątku POSIX. Wyraźna różnica zaczyna być widoczna już przy 1 kHz, kiedy to wątek POSIX jest opóźniony względem wartości zadanej o więcej niż 5% (*Rysunek 20*). Rezultaty otrzymane dla wątku Xenomai są porównywalne z wynikami uzyskanymi dla Hard-PWM, co oznacza, że PWM programowy jest w stanie działać równie dobrze i wydajnie jak PWM sprzętowy.

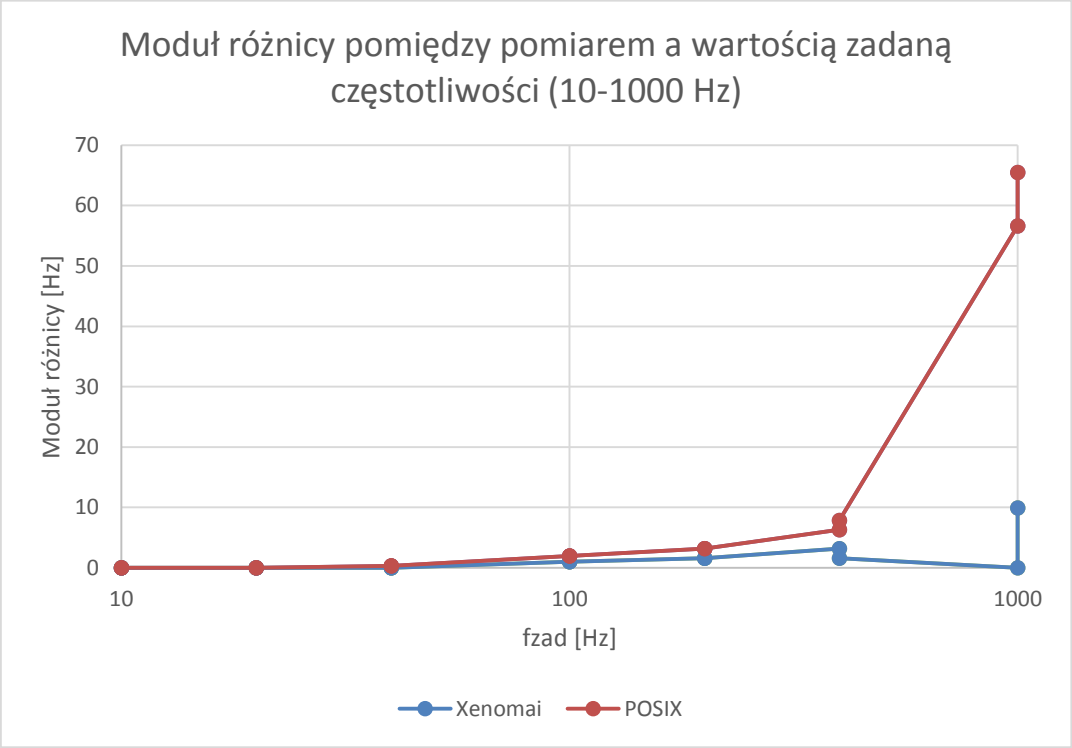


Rysunek 20. Przebieg sygnału programowego PWM dla częstotliwości 1000 Hz (Niebieski – Xenomai, Czerwony – POSIX).

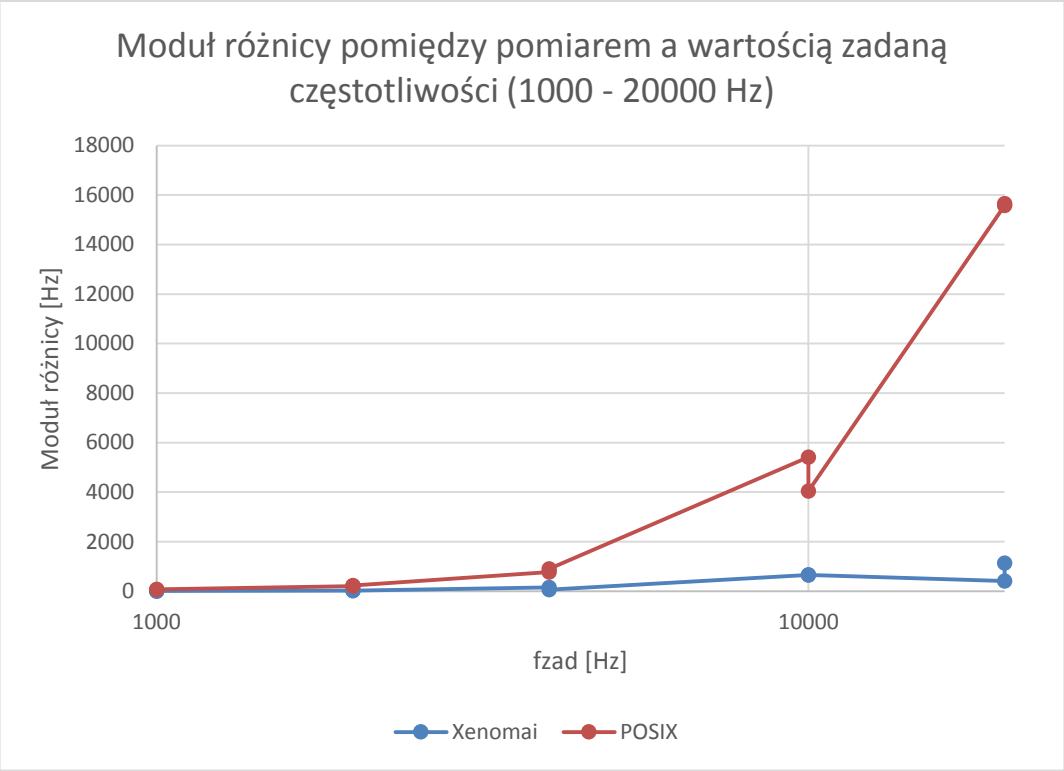
Z otrzymanych pomiarów wynika, iż wyniki uzyskane dla PWMa działającego w wątku czasu rzeczywistego, są lepsze od PWMa działającego na POSIXowych wątkach. Na *Wykresach 3-5* przedstawione zostało porównanie różnic pomiędzy wartościami częstotliwości oraz okresu uzyskanymi a zadanymi dla programowego PWM działającego w jądrze Xenomai i standardowych w jądrze systemu.



Wykres 3. Moduł różnicy pomiędzy pomiarem a wartością zadaną okresu.



Wykres 4. Moduł różnicy pomiędzy pomiarem a wartością zadaną częstotliwości w zakresie 10-1000 Hz.



Wykres 5. Moduł różnicy pomiędzy pomiarem a wartością zadaną częstotliwości w zakresie 1-20 kHz.

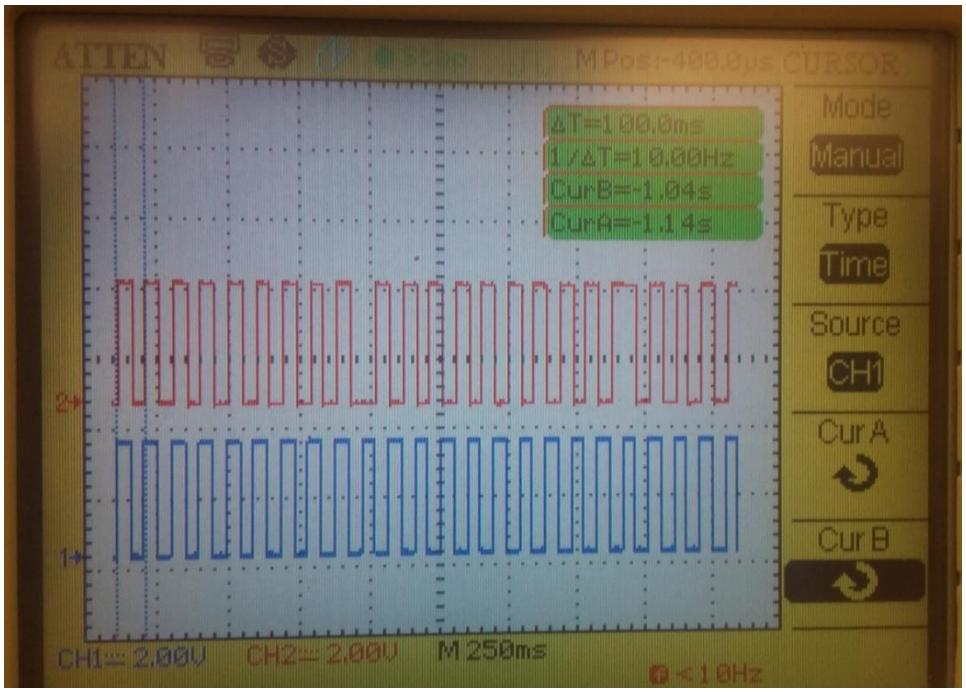
Na powyższych wykresach jednoznacznie widać, że PWM działający w wątku RT stworzonym przy pomocy Xenomai-API działa zdecydowanie lepiej od standardowego wątku non-RT. Przy wysokich częstotliwościach wątek non-RT znacząco odbiega od wartości zadanych i tym samym przestaje spełniać swoją funkcję. W praktycznym zastosowaniu spowodowałoby to znaczące opóźnienia w działaniu, a tym samym mogłoby spowodować dużą utratę precyzji urządzenia wykorzystującego taki PWM. Natomiast wątek RT praktycznie w całym zakresie częstotliwości był w stanie spełniać wymogi wartości zadanych, a tym samym jego przydatność w praktycznym zastosowaniu jest bardzo duża.

Ostatnim badaniem, które ostatecznie miało pokazać różnicę pomiędzy działaniem wątków RT a non-RT, było badanie przebiegu fali PWM w warunkach z obciążeniem

procesora. W tym celu do działania aplikacji włączone zostały wątki obciążające procesor. Zadanie te utworzone zostały w wątkach non-RT z niskim priorytetem, zaś wątki do sterowania PWM pozostałby bez zmian. Pomiary oscyloskopem przeprowadzone zostały dla trzech wartości częstotliwości:

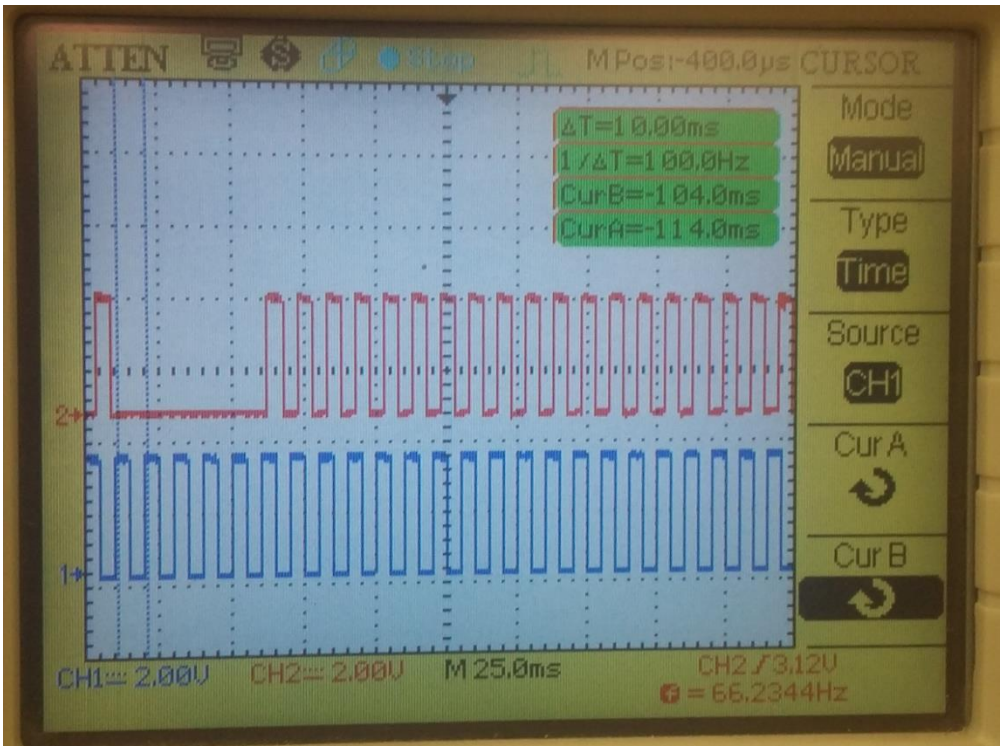
- 10 Hz
- 100 Hz
- 1 kHz

W przypadku częstotliwości zadanej 10 Hz wpływ działania wątków obciążających na system był znikomy, jednakże można było zaobserwować pewne opóźnienia w działaniu PWM non-RT (pojedyncze okresy o wydłużonym czasie trwania), co przedstawia przebieg czerwoną linią na *Rysunku 21*. W tych samych warunkach w przebiegu PWM RT (niebieski), nie zaobserwowano żadnych opóźnień w działaniu.



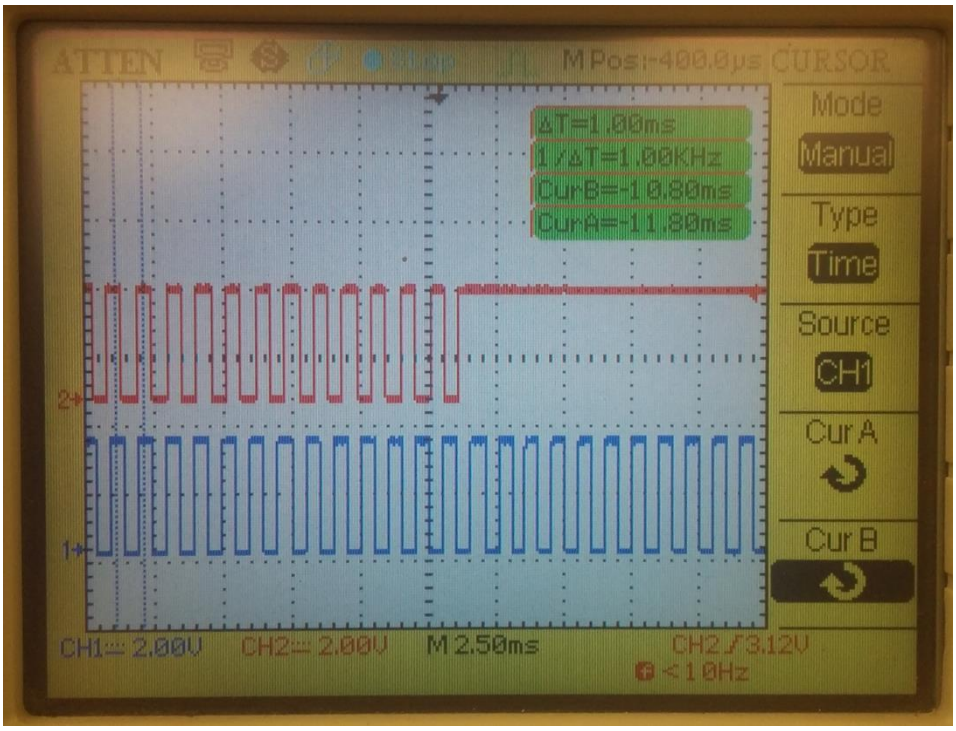
Rysunek 21. Przebieg fali PWM dla wątków RT i non-RT w warunkach z obciążeniem systemu dla częstotliwości 10 Hz.

Przy wartości częstotliwości zadanej 100 Hz różnica w działaniu obu PWM, zaczęła być wyraźnie widoczna, co przedstawia *Rysunek 22*. Przy tej częstotliwości wątek non-RT zdecydowanie nie był w stanie dotrzymać precyzji działania i co jakiś czas musiał być wstrzymany, tak aby procesor mógł obsłużyć działanie innych wątków w systemie, podczas gdy wątek RT był w stanie nadążać z działaniem i dotrzymać spełnianie wartości zadanych.



Rysunek 22. Przebieg fali PWM dla wątków RT i non-RT w warunkach z obciążeniem systemu dla częstotliwości 100 Hz.

W przypadku gdy częstotliwość pracy PWM została zwiększona do 1000 Hz, wątek non-RT do obsługi PWM zostawał zatrzymywany na znacząco długie okresy działania PWM, w celu obsługi przez system pozostałych wątków. Przy takiej częstotliwości pracy PWM w wątku non-RT był praktycznie bezużyteczny ponieważ, przez około połowę swojej pracy był wstrzymany. Natomiast wątek RT do obsługi PWM nawet przy tej częstotliwości działał bez zarzutu i wciąż był w stanie wykonywać swoje działanie (*Rysunek 23*).



Rysunek 23. Przebieg fali PWM dla wątków RT i non-RT w warunkach z obciążeniem systemu dla częstotliwości 1 kHz.

Rozdział 6.
Podsumowanie

Konfiguracja systemu czasu rzeczywistego jest złożonym i skomplikowanym procesem, na który składa się wiele czynników. Każdy budowany system, musi być przystosowany do działania na konkretnym, dedykowanym sprzęcie, gdyż tylko w ten sposób można zapewnić jego wysoką wydajność. W przypadku nieskomplikowanych aplikacji korzyści płynące z zastosowania systemu czasu rzeczywistego nie zawsze mogą być tak widoczne, jak to ma miejsce w bardziej złożonych projektach. Kluczowe jest precyzyjne określenie wymagań jakie stawiamy budowanemu systemowi oraz przeanalizowanie wszystkich czynników oraz parametrów mających wpływ na działanie systemu.

Badania przeprowadzone w ramach tego projektu pokazały jak dużo korzyści niesie za sobą wykorzystanie systemu czasu rzeczywistego. Osiągnięte rezultaty pokazują, że wykorzystanie aplikacji działających w takim systemie, może konkurować z dedykowanymi sprzętowymi układami. W trakcie badań pokazane zostało jak wątki czasu rzeczywistego przewyższają zwykłe wątki systemowe w czasie działania oraz dotrzymywaniu realizacji powierzonych zadań. W przypadku złożonych aplikacji, w których działa wiele procesów o różnych priorytetach, niekiedy decydujące może być użycie wątków RT w celu zapewnienia gwarancji poprawnego działania kluczowych procesów systemu.

Skonfigurowanie systemu czasu rzeczywistego na bazie Xenomai daje ogromne możliwości, oraz pozwala na projektowanie złożonych aplikacji o wydajniejszym działaniu od standardowych systemów. Korzyści płynące z zastosowanie jądra Xenomai obrazuje Xenomai-API, które w swoim interfejsie daje ogromne możliwości projektantowi systemu, pozwalające na łatwe i intuicyjne tworzenie złożonych aplikacji. Wyniki badań zamieszczone w publikacji

Jeremy H. Browna pod tytułem “How fast is fast enough? Choosing between Xenomai and Linux for real-time applications” [5], pokazują jak znaczącą poprawę działania systemu można uzyskać dzięki zastosowaniu Xenomai (*Rysunek 24*).

Config	Experiment	# samples	Median	95%	100%
stock	linux-chrt-user	1840823	67μs	307μs	17227μs
rt	linux-chrt-user	1849438	99μs	157μs	796μs
xeno	xeno-user	1926157	26μs	59μs	90μs
stock	linux-kernel	1259410	7μs	16μs	597μs
rt	linux-kernel	1924955	28μs	43μs	336μs
xeno	xeno-kernel	1943258	9μs	18μs	37μs

Table 4: cross-configuration response experiments: latency from input GPIO change to corresponding output GPIO change.

Config	Experiment	95% period	100% period
stock	linux-chrt-user	1.63 kHz	0.03 kHz
rt	linux-chrt-user	3.18 kHz	0.63 kHz
xeno	xeno-user	8.47 kHz	5.56 kHz
stock	linux-kernel	31.25 kHz	0.84 kHz
rt	linux-kernel	11.63 kHz	1.49 kHz
xeno	xeno-kernel	27.78 kHz	13.51 kHz

Table 5: cross-configuration response experiments: approximate highest frequency possible for which latency does not exceed 1/2 period, for 95% and 100% cases.

Rysunek 24. Wyniki badań Jeremy H. Browna [5]

Z przeprowadzonych badań wynika, że w przypadku odpowiedniego użycia Xenomai działającego w przestrzeni użytkownika, możliwe są do uzyskania częstotliwości odpowiedzi systemu powyżej 5 kHz, gdy w tych samych warunkach standardowy system osiąga wyniki o 3 rzędy wielkości mniejsze, tak samo jest w przypadku badań opóźnienia systemu. Nie można wyników tych badań bezpośrednio przyrównać do przeprowadzonych w ramach tego projektu gdyż sposób pomiaru tych wartości oraz warunki w jakich się to odbywało znacząco się od siebie różniły, Brown do sterowania wartością pinu stworzył dodatkowy moduł działający w przestrzeni jądra systemu, co znacząco wpłynęło na osiągane rezultaty. Nie mniej jednak badania te pokazują jaki potencjał jest w systemach tworzonych na bazie Xenomai. Można się spodziewać, że komercyjne użycie nakładki Xenomai będzie coraz powszechniejsze.

Właściwa konfiguracja systemu jest dość skomplikowana i stworzenie systemu, który będzie znacząco wydajniejszy wymaga ogromnej pracy oraz szczegółowej wiedzy dotyczącej działania systemu operacyjnego. Badania przeprowadzone i zamieszczone w artykule pod tytułem „Precise PWMs with GPIO using Xenomai kernel module" przez Andreya Nech'a [*23*] pokazują, że niekiedy proste użycie Xenomai nie daje wystarczających rezultatów, choć poprawia osiągane wyniki.

Bibliografia

Literatura:

[1] Majdzik P. „*Programowanie współbieżne*” Wydawnictwo Helion, 2012

[2] Praca zbiorowa pod redakcją Andrzeja Karbowskiego i Ewy Niewiadomskiej-Szynkiewicz. „*Programowanie równoległe i rozproszone*” Oficyna wydawnicza Politechniki Warszawskiej, Warszawa 2009

[3] Upton E, Halfacree G. „Raspberry Pi Przewodnik użytkownika” Wydawnictow Helion, 2013

Artykuły i Publikacje:

[4] Marchewka D. „Stosowanie systemów wbudownaych do sterowania robotami moblilnymi” Automatyka 2008 Tom 12 Zeszyt 2

[5] Dr. Jeremy H. Brown “How fast is fast enough? Choosing between Xenomai and Linux for real-time applications”

Strony internetowe (dostęp 30.08.2016):

[6] http://sequoia.ict.pwr.wroc.pl/~witold/scrsk/

[7] http://xenomai.org/

[8] https://pl.wikipedia.org/wiki/System_operacyjny

[9] http://xenomai.org/documentation/xenomai-2.4/html/api/group__native.html

[10] https://xenomai.org/introducing-xenomai-3/

[11] http://students.mimuw.edu.pl/SO/Projekt03-04/temat4-g2/RTAI.pdf

[12] http://igm.univ-mlv.fr/~masson/v2/Teachings/IMC-4201C/RTAI/rtai.pdf

[13] http://www.slideshare.net/nylon7/realtime-soultion

[14] https://botland.com.pl/moduly-glowne-i-zestawy-raspberry-pi/972-raspberry-pi-model-b-512mb-ram.html

[15] http://www.instructables.com/id/Simple-and-intuitive-web-interface-for-your-Raspbe/

[16] http://hertaville.com/rpipwm.html

[17] https://github.com/kinsamanka/picnc/blob/wiki/RPiXenomaiKernel.md

[18] https://github.com/awesomebytes/xenorasp

[19] https://buildroot.org/

[20] https://delog.wordpress.com/2014/10/10/wireless-on-raspberry-pi-with-buildroot/

[21] http://jbohren.com/articles/xenomai-precise/

[22] http://www.armadeus.org/wiki/index.php?title=Xenomai

[23] http://letsmakerobots.com/node/32347

Zawartość płyty CD:

- Praca dyplomowa w postaci źródłowej (doc)
- Praca dyplomowa w postaci pliku pdf
- Kod źródłowy oprogramowania
- Wersja skompilowana programu
- Obraz stworzonego systemu

Konfiguracja systemu czasu rzeczywistego na bazie Xenomai/RTAI

Streszczenie

Głownym celem tego projektu było stworzenie systemu czasu rzeczywistego na bazie Xenomai lub RTAI, w celu pokazania przewagi tego typu systemów nad standardowymi. Spodziewanym rezultatem było otrzymanie wydajniejszego oraz precyzyjniejszego pod względem realizacji ograniczeń czasowych dla zadanych procesów.

W celu spełnienia tych wymagań, użyty został system wbudowany (Raspberry PI). W pierwszej kolejności przetestowany został system bazowy, nie będący systemem czasu rzeczywistego. Do badań użyty został system operacyjny Rasbian. Do pomiarów przygotowana została specjalna aplikacja, mająca na celu generowanie programowo fali PWM, przy użyciu wątków systemowych POSIX, oraz wątków mających na celu obciążenie systemu.

Kolejnym etapem badań, było stworzenie nowego systemu bezującego na Linux, z dodatkowym modułem Xenomai. Nowo przygotowany system został przetestowany, zmodyfikowaną aplikacją. Do aplikacji został dodany wątek czasu rzeczywistego, którego celem było generowanie fali PWM. Oba wątki (czasu rzeczywistego oraz nie czasu rzeczywistego) zostały porównane.

W uzyskanych wynikach zostało udowodnione, że wątki czasu rzeczywistego działają wydajniej oraz stabilniej od wątków standardowych systemu. Badanie przebiegu fali PWM pokazały, że wątki czasu rzeczywistego do generowania fali PWM, mogą osiągać porównywalne rezultaty do sprzetowych dedykowanych układów do generowania fali PWM.

Real time system based on Xenomai/RTAI

Abstract

The main purpose of this project was to create a real time system based on Xenomai or RTAI patch to show advantages of real time system comparing to standard systems. It was expected to see more efficient system precisely realizing time constraints of given tasks.

To achieve this requirement, a build-in system has been used, which is call Raspberry PI. At the beginning, a base system (non_RT) has been measured. The base operational system (Rasbian) has been installed on device. Then a special application has been prepared. Main goal for application was to create POSIX threads in which dedicated tasks were running, such as PWM control task and other threads just to overload the system.

Next step of the project was to prepare a new operational system based on Linux with special addition of real time module call Xenomai. After new system has been prepared, new tests has been made. Application has been changed to include new real time task for controlling PWM. Two threads controlling PWM (real time thread and non-real time thread) have been compared.

In obtained results it has been proved that real time threads are more efficient and stable than non-real time threads. The measurements show that PWM in real time task can achieve frequency very close to the one that build-in (hardware) dedicated device can achieve.