Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Rafał Zuzak

kierunek studiów: **fizyka techniczna**

specjalność: **fizyka jądrowa**

Budowa i oprogramowanie automatycznych systemów pomiarowych w elektronice jądrowej

Promotor: **dr hab. Marek Idzik**

Kraków, lipiec 2009

Serdecznie dziękuję za poswięcony mi czas, przekazaną wiedzę, a także za dużą pomoc przy tworzeniu pracy, mojemu promotorowi Panu dr hab. Markowi Idzikowi

autor

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Kraków, 26 czerwca 2009

Tematyka pracy magisterskiej i praktyki dyplomowej Rafała Zuzaka, studenta V roku studiów kierunku fizyka techniczna, specjalności fizyka jądrowa

Temat pracy magisterskiej: **Budowa i oprogramowanie automatycznych systemów pomiarowych w elektronice jądrowej**

Opiekun pracy: dr hab. Marek Idzik
Recenzenci pracy: ....
Miejsce praktyki dyplomowej: WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Praktyka dyplomowa:
   — zapoznanie się z ideą...,
   — przygotwanie oprogramowania...,
   — dyskusja i analiza wyników...
   — sporządzenie sprawozdania z praktyki.
4. Kontynuacja tworzenia oprogramowania związanego z tematem pracy magisterskiej.
5. Zebranie i opracowanie dokumantacji dla oprogramowania.
6. Analiza wyników pracy, ich omówienie i zatwierdzenie przez opiekuna.
7. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: 26 czerwca 2009

....
(podpis kierownika katedry) (podpis opiekuna)

Merytoryczna ocena pracy przez opiekuna:

Końcowa ocena pracy przez opiekuna: ....

Data: .... Podpis: ....

Merytoryczna ocena pracy przez recenzenta:

Końcowa ocena pracy przez recenzenta: ....

Data: .... Podpis: ....

Skala ocen: (6.0 – celująca), 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostatetczna, 2.0 – niedostatetczna

Spis treści

Wstęp . . . 11

Rozdział 1. Budowa i działanie oprogramowania dla urządzeń pomiarowych . . . 13

1.1. Standard GP-IB (*General Purpose-Interface Bus*) . . . 13
1.2. Budowa oprogramowania dla urządzeń GP-IB . . . 17
1.3. Urządzenia dla których stworzone zostało oprogramowanie . . . 22
1.3.1. Analizator widma HP4195A . . . 22
1.3.2. Analizator widma HP4395A . . . 23
1.3.3. Analizator urządzeń półprzewodnikowych B1500A . . . 24
1.3.4. Miernik wartości LCR, HP4284A . . . 26
1.3.5. Analizator urządzeń półprzewodnikowych HP4145B . . . 27
1.3.6. Zasilacz HP6624A . . . 28
1.3.7. Generator SMB100 . . . 28
1.3.8. Generator AWG2021 . . . 29
1.3.9. Generator AFG3102 . . . 29
1.3.10. Oscyloskopy serii TDS . . . 30

Rozdział 2. Przegląd koncepcji dotyczących projektowanego środowiska GUI . . . 33

2.1. Środowisko graficzne na bazie biblioteki GTK 3.0 języka C++ . . . 33
2.2. Środowisko graficzne oparte o technologie internetowe . . . 34
2.3. Środowisko tworzone przy wykorzystaniu koncepcji programowania z użyciem aktorów 34
2.3.1. Środowisko graficzne Diva . . . 35
2.3.2. Środowisko pomiarowe Ptolemeusz . . . 35
2.4. Podsumowanie rozważanych koncepcji . . . 36

Rozdział 3. Aktor jako pojęcie w programowaniu . . . 37

3.1. Prace Carla Hewitta i Gul Agha . . . 37
3.1.1. Programowanie współbieżne . . . 39
3.2. Różnice i podobieństwa pomiędzy obiektem, a aktorem . . . 40

Rozdział 4. Środowisko GUI . . . 41

4.1. Ogólna struktura środowiska graficznego . . . 41
4.1.1. Pliki konfiguracyjne. . . . 42

4.2. Informacje na temat kodu źródłowego, najważniejsze klasy, metody i typy 44
4.2.1. Parametry, Port-Parametry i Porty 44
4.2.2. Metody prefire(), fire(), postfire() i wrapup() 45
4.3. Główne elementy środowiska graficznego 46
4.3.1. Uruchamianie i okno powitalne 46
4.3.2. Okno tworzenia modelu. Opis elementów interfejsu 47
4.4. Tworzenie własnych elementów 48
4.4.1. Tworzenie aktora poprzez nowy plik Java 49
4.4.2. Tworzenie aktora poprzez edytor graficzny z aktorów już istniejących 56
4.4.3. Reżyser 58
4.5. Tworzenie modelu 59

Rozdział 5. Przykładowe modele pomiarowe 63

5.1. Stanowisko pomiarowe do pomiarów statycznych i pomiarów mocy układów DAC . . . 63
5.1.1. Budowa i oprogramowanie stanowiska 63
5.1.2. Model wykorzystywany w pomiarach 65
5.1.3. Wyniki 67
5.2. Pomiary I-V i C-V krzemowych sensorów firmy Hamamatsu 69
5.2.1. Model do pomiarów I-V krzemowych sensorów firmy Hammamatsu 70
5.2.2. Wyniki 72
5.2.3. Model do pomiarów C-V krzemowych sensorów firmy Hammamatsu 72
5.2.4. Wyniki 76
5.3. Badanie generatorów w celu określenia ich parametrów 77
5.3.1. Procedura Pomiarowa 78
5.3.2. Model wykorzystywany w pomiarach 80
5.3.3. Pomiary dla generatora AWG2021 81
5.3.4. Pomiary dla generatora AFG3102 84
5.3.5. Pomiary dla generatora SMB100 87
5.3.6. Wnioski 91
5.4. Stanowisko do pomiarów szumów układu elektroniki Front-End 91
5.4.1. Procedury pomiarowe dla zaprojektowanego stanowiska 92
5.4.2. Model wykorzystywany w pomiarach 92

Podsumowanie 93

Dodatek A. Instalacja libgpib 95

Dodatek B. Biblioteka Stringutils 97

Dodatek C. Licencja 101

Dodatek D. Struktura katalogów 103

Bibliografia 105

Spis rysunków 107

Spis tablic 111

Wstęp

Celem pracy było stworzenie oprogramowania sterującego urządzeniami pomiarowymi i automatyzacja pomiarów wykonywanych w Zespole Elektroniki i Detekcji Cząstek. W pracy poprzez automatyczny system pomiarowy rozumie się zestaw urządzeń pomiarowych, przeznaczonych do określonego pomiaru, którego wykorzystanie ma wymagać jak najmniejszej interakcji człowieka. Zespół Elektroniki i Detekcji Cząstek zajmuje się budową systemów detekcji, a także projektowaniem układów elektroniki odczytu dla potrzeb eksperymentów fizyki wysokich energii. Projektowane układy często wymagają złożonych pomiarów, w których istnieje konieczność powtarzania danego pomiaru/czynności bardzo dużą ilość razy. Dlatego też istanieje potrzeba zautomatyzowania pomiarów, aby przyspieszyć cały proces.

Wszystkie stosowane urządzenia pomiarowe do komunikacji wykorzystują standard GP-IB (z ang. *General Purpose - Interface Bus*). W celu komunikacji pomiędzy urządzeniem, a komputerem PC używano darmowej biblioteki libgpib. Biblioteka ta dedykowana jest dla języka programowania C++. Dlatego też wszystkie programy sterujące tworzone były w tym języku. Pierwotnie oprogramowanie było oprogramowaniem konsolowym. Łączenie kilku programów sterujących (czyli urządzeń pomiarowych), odbywało się za pomocą skryptów systemowych. Aby jeszcze bardziej ułatwić tworzenie zautoamtyzowanych systemów pomiarowych, autor zaproponował stworzenie środowiska graficznego, którego zadaniem miało być proste tworzenie stanowisk pomiarowych. Środowisko to w zamyśle autora miało korzystać ze stworzonych już programów sterujących tak, aby możliwy był dwutorowy rozwój oprogramowania, tzn. poprzez skrypty i poprzez środowisko graficzne.

Praca składa się z pięciu rozdziałów. Poza tym umieszczono w pracy szereg załączników, których celem jest przybliżenie aspektów technicznych stworzonego oprogramowania.

W rozdziale pierwszym opisany jest standard GP-IB. Jest to standard transmisji danych wykorzystywany przez wszystkie urządzenia opisane w pracy. Opisany jest sam standard, jak również metoda tworzenia sterowników wykorzystywana przy tworzeniu oprogramowania w labolatorium Zespołu Elektroniki i Detekcji Cząstek. W rozdziale tym opisane są także wszystkie programy sterujące dla wykorzystywanych urządzeń.

Rozdział drugi zawiera krótki przegląd koncepcji, które brane były pod uwagę jako możliwe do wykorzystania podczas tworzenia środowiska graficznego GUI. Starano się przedsatawić krótko zalety i wady kolejnych rozwiązań.

W rozdziale trzecim przedstawiono teorię programowania orientowanego aktorowo. Z racji pewnych różnic wobec, na przykład programowania orientowanego obiektowo, uznano, iż rozdział taki jest konieczny. W tym rozdziale zostały przedstawione prace, które dały początek tej filozofii programowania. W rozdziale tym zostało także wspomniane programowanie współbierzne,

z którym programy orientowane na aktorów radzą sobie bardzo dobrze.

Kolejny, czwarty rozdział, przedstawia opis środowiska graficznego stworzonego dla labolatorium Zespołu Elektroniki i Detekcji Cząstek. Opisano główne założenia, oraz technologię z jakiej korzysta interfejs graficzny, czyli narzędzie Ptolemeusz. Zaprezentowane tu zostały wszystkie najważniejsze opcje środowiska, wraz z przykładami. Przedstawiony został także sposób tworzenia aktorów, co jest najważniejszą cechą przy korzystaniu ze środowiska. Zaprezentowano także tworzenie przykładowego modelu pomiarowego, co ma rozjaśniać znaczenie funkcji dostępnych poprzez interfejs graficzny.

Rozdział piąty prezentuje stanowiska pomiarowe, które udało się wykonać przy pomocy napisanych sterowników i środowiska graficznego. Stworzone zostały cztery takie stanowiska. Pierwsze służy do badania parametrów układu prztwornika cyfrowo-analogowego (DAC) zaprojektowanego w Zespole Elektroniki i Detekcji Cząstek. Drugie stanowisko wykorzystywane jest do badania parametrów generatorów sygnału, celem wybrania najlepszego, który następnie ma być wykorzystywany w pomiarach układu prztewornika analogowo-cyfrowego (ADC), także zaprojektowanego w Zespole Elektroniki i Detekcji Cząstek. Kolejne stanowisko umożliwia określenie parametrów sensorów krzemowych firmy Hammatsu i porównanie ich z wartościami dostarczonymi przez producenta. Ostatnie stanowisko służy do określenia parametrów szumowych układu elektroniki Front-End zaprojektowanego również w Zespole Elektroniki i Detekcji cząstek. Do wszystkich powyższych pomiarów użyto szeregu urządzeń, które pokrótce opisane są w rozdziale drugim.

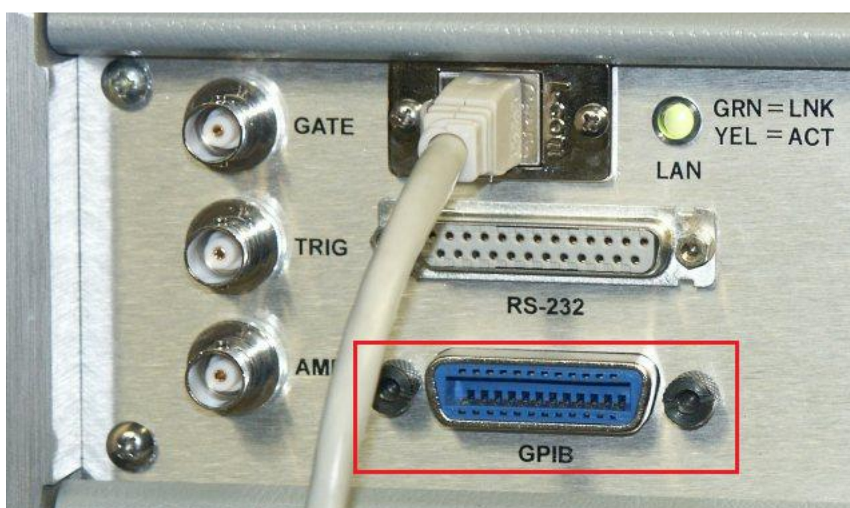
Rozdział 1

Budowa i działanie oprogramowania dla urządzeń pomiarowych

Podstawą każdego omawianego dalej systemu pomiarowego jest sterownik (dalej zwany także programem sterującym) zarządzający danym urządzeniem. Jest to program bezpośrednio komunikujący się z urządzenie. Sterownik jest programem działającym najbliżej sprzętu i to on przesyła i odbiera informacje z urządzenia. Z racji istotności tego zagadnienia, w rozdziale tym przedstawiono sposób tworzenia owych sterowników. Na początku rozdziału zostaną przybliżone pojęcia takie jak standard GP-IB (z ang. *General Purpose - Interface Bus*), co jest niezbędne, aby dokładnie zrozumieć sposób budowania oprogramowania, gdyż wszystkie wykorzystane tu urządzenia korzystają z tego właśnie standardu. Opisane zostaną jego podstawy fizyczne jak i programowe.

1.1. Standard GP-IB (*General Purpose-Interface Bus*)

Standard GP-IB stworzony został głównie w celu komunikacji pomiędzy urządzeniami pomiarowymi. Zaprojektowany został pierwotnie przez firmę Hewllet-Packard, a następnie po kilkunastu latach stworzona została oficjalna jego wersja pod nadzorem IEEE (dlatego inna nazwa tego standardu to IEEE 488.1 i IEEE 488.2)[1]. Wszystkie urządzenia wspierające omawiany standard posiadają port GP-IB. Port taki widoczny jest na rysunku 1.1.



Rysunek 1.1. *Port GP-IB zaznaczony ramką na czerwono.*

Wspomnieć należy, iż starsze urządzenia zamiast portu typu GP-IB, mogą posiadać port oznaczony jako HP-IB (z ang. *Hewllet Packard - Interface Bus*), który jest wspomnianą starszą jego wersją rozwijaną jeszcze przez firmę Hewllet-Packard. Podczas prac nad oprogramowaniem natknięto się na oba rodzaje portów, jednakże obie wersje standardu do komunikacji wyko-

rzystują ten sam rodzaj przewodu, co nie powoduje konieczności stosowania innego medium przesyłowego. Mogą jednakże występować różnice w rozpoznawanych komendach, na co należy zwracać uwagę posługując się starszymi urządzeniami (przeważnie występujące różnice opisane były w podręcznikach dla danego urzadzenia).

W standardzie jako medium przesyłowe wykorzystuje się 24 żyłowy przewód, którego budowę przedstawiono na rysunku 1.2.

Rysunek 1.2. *Z lewej: Schematyczny widok na przekrój przewodu. Z prawej: Przewód GP-IB*

W tabeli 1.1 zamieszczono krótki opis linii sterujących przedstawionych na rysunku 2.2 z lewej:

nazwa komunikatu	tryb pracy	opis komunikatu
NRFD	Listner	Żądanie dodatkowych informacji od urządzenia, np. zakresu etc.
NDAC	Listner	potwierdzenie odebrania danych i ew. zgłoszenie gotowości przyjęcia kolejnych
DAV	Talker	urządzenie nadające zawiadamia, że są dostępne do odebrania dane na liniach DIOx
ATN	Controller	aktywny wtedy gdy wysyłane są komunikaty przez kontroler
EOI	Talker	używany jako zakończenie polecenia
IFC	Controller	kontroler zeruje informacje o adresach
REN	Controller	aktywowanie, powoduje że kontrole nad daną transmisją przejmuje adresat
SRQ	Talker	informacja urządzenia o tym iż wymaga obsłużenia(odebrania danych etc.)

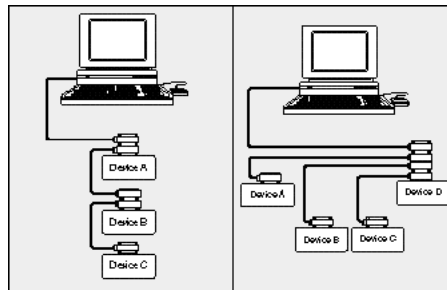
Tablica 1.1. *Tabela przedstawiająca linie sterujące w standardzie GP-IB. Pierwsza kolumna - nazwa rozkazu, druga - tryb pracy urządzenia do jakiego jest wysyłany, trzecia - krótki opis.*

Oprócz linii sterujących na schemacie widoczne są linie oznaczone jako DIOx (gdzie x to liczba z zakresu od 1 do 8), są to linie transmisji danych. Istnieje także szereg linii oznaczonych jako GND, ich głównym zadaniem jest eliminacja zakłóceń (poprzez ekranowanie), które powstawać mogą w przewodzie podczas pracy.

Z uwagi na występujące zakłócenia, suma długości przewodów w sieci ograniczona jest do 20 metrów na całą sieć. Natomiast długość przewodu pomiędzy danymi dwoma urządzeniami

nie powinna przekraczać 2 metrów. Wspomnieć także należy, iż prędkość przesyłania informacji ograniczona jest do 1 MB/s (w najnowszej wersji standardu IEEE 488.2 możliwa jest praca z większą prędkością, ale zaleca się wspomniane ograniczenie celem wyeliminowania dodatkowych strat informacji).

Sieć urządzeń budowana może być w sposób liniowy, lub w tak zwaną gwiazdę, co przedstawiono na rysunku 1.3.



Rysunek 1.3. *Sieć urządzeń może być tworzona w sposób liniowy(lewy obrazek), lub w gwiazdę (prawy obrazek).*

Urządzenia pracować mogą w trzech trybach, zwanych *Listner*, *Talker* lub *Controller*. Jak nie trudno się domyśleć, tryb pracy *Listner* to nasłuchiwanie rozkazów od kontrolera, natomiast *Talker* to przekazywanie informacji na żądanie kontrolera. Kontroler sprawuje rolę nadzorcy, to poprzez niego urządzenie komunikuje się na przykład z komputerem PC. Wspomnieć należy, iż każde urządzenie w sieci posiada swój unikalny adres. Jeden kontroler obsługiwać może do trzydziestu urządzeń z adresami z przedziału 1 do 31. Aby zbudować sieć z większą liczbą urządzeń konieczne jest zastosowanie większej liczby kontrolerów i połączenie ich w sieć. W laboratorium wykorzystywane były zewnętrzne kontrolery firmy *National Instruments* pracujące jako urządzenia [GP-IB]-[USB] podłączane do komputera PC. Wygląd takiego kontrolera zaprezentowano na rysunku 1.4.

Rysunek 1.4. *Kontroler GP-IB stosowany w laboratorium. Do komputera PC podłączany za pomocą portu USB.*

Do komunikowania się z urządzeniami kontroler wykorzystuje szereg standardowych komend, które urządzenia z nim współpracujące muszą obsługiwać. Każda komenda przekazywana jest

do urządzenia w formie kodu ASCII (7 bitowego). Dodatkowo każde urządzenie posiada zbiór własnych komend, odpowiednich dla typu urządzenia (na przykład woltomierz posiada komendy pozwalające odczytać napięcie etc.). Komendy takie tworzy producent urządzenia za pomocą ścisle określonych w normie IEEE 488.2 narzędzi.

Komenda	opis komendy
*IDN?	Żądanie podania informacji o urządzeniu
*RST	Reset urządzenia
*CLS	Wyzerowanie bajtu stanu urządzenia
*RST	Reset urządzenia
*OPC?/*OPC	Pobranie stanu urządzenia/Sygnał zakończenia operacji
*TST?	Wykonanie testu urządzenia
*WAI	Wstrzymanie pracy urządzenia
*ESR?	Pobranie bajtu stanu urządzenia
*ESE/*ESE?	Ustawienie maski stanu/pobranie maski stanu
*RST	Reset urządzenia
*SRE/*SRE?	Ustawienie maski SRE/pobranie maski SRE
*STB?	Pobranie bajtu stanu STB

Tablica 1.2. *Zbiór standardowych komend według standardu IEEE 488.2, które wspierać muszą wszystkie urządzenia z nim zgodne.*

W tabeli 1.2 wprowadzono kilka pojęć, które wymagają wyjaśnienia. Chodzi o bajty i maski stanu, i związane z nimi rejestry o nazwach kolejno ESR, ESE, SRE i STB. Rejestr ESR (z ang. *Event Status Register*) służy przechowywaniu informacji o wykonanych komendach. Jest to rejestr 8 bitowy z tym, iż bit numer 6 nie jest używany. Jest to bit, który może być wykorzystany przez osobę programującą do innych celów. Rejestr ESE (z ang. *Event Status Enable*) służy kontroli zdarzeń które zaszły w rejestrze ESR, pozwala na definiowanie odpowiednich wartości na każdym z dostępnych 8 bitów. Kolejno istnieje rejestr STB (z ang. *Status Byte Register*) w którym przechowywane są opdpwiednie informacje, przykładowo w bicie numer 4 przechowywany jest bit nazwany MAV, który podaje informacje o dostępności danych do odczytania. W bicie numer 5 rejestru STB przechowywane są informacje jakie programista uzyskuje poprzez przeprowadzenie sumy logicznej wybranych bitów z rejestru ESR i ESE. Bit numer 6 jest zależny od trybu pracy urządzenia, gdy jest to tryb *Poll*, to bit ten zawsze ma wartość 0. W innej sytuacji generowany jest sygnał nazwany SRQ, który jest aktywny gdy zajdzie odpowiednie zdarzenie w ESR względem ESE, lub gdy w kolejce danych pojawią się informacje do oczytania. Bity numer 0,1,2,3 i 7 są bitami wolnymi, które użytkownik (programista) może wykorzystywać do innych celów.

Dodatkowo na początku lat 90-tych wprowadzono możliwość tworzenia nowych komend (o czym wspomniano już wyżej) za pomocą języka SCPI (z ang. *Standard Commands for Programmable Instruments*). Pozwoliło to twórcowm sprzętu w łatwy sposób implementować odpowiednie komendy, co bardzo mocno spopularyzowało omawiany standard. Dzięki temu, ogromna większość sprzętu pomiarowego obecnie produkowanego posiada wsparcie technologii GP-IB. Oczywiście, jak było już wspomniane, proces tworzenia komend jest dokładnie opisany w normie IEEE 488.2.

Podsumowując standard GP-IB w łatwy sposób pozwala łączyć ze sobą większą liczbę urządzeń pomiarowych. Budowanie stanowiska pomiarowego sprowadza się do fizycznego połączenia urządzeń, wpięcia kontrolera i posiadania odpowiedniego oprogramowania. Istnieje duża swoboda konfiguracji urządzeń, nowe można podpinać „w locie" (czyli bez konieczności wyłączania pozostałych urządzeń), co także nie jest bez znaczenia. Pozwala na jeszcze bardziej płynne umieszczanie nowych elementów projektowanego systemu pomiarowego.

1.2. Budowa oprogramowania dla urządzeń GP-IB

Oprogramowanie pisane było w języku C++ z wykorzystaniem darmowej biblioteki *libgpib*.[2] Dlatego też do skompilowania oprogramowania wymagane jest zainstalowanie tej biblioteki w systemie. Sposób instalacji biblioteki przedstawiono w załączniku A. Aby ułatwić tworzenie oprogramowania sterującego urządzeniami, stworzona została klasa, która posiada metody ułatwiające komunikacje z urządzeniami. Klasą tą jest *GpibDev*(plik *gpibdev.h* i *gpibdev.cpp*). Tworzy ona z funkcji zawartych w bibliotece *libgpib* metody w znaczny sposób ułatwiające budowanie programów sterujących. Czterema podstawowymi metodami z klasy *GpibDev* są te przedstawione w tabeli 1.3.

nazwa metody	opis metody
check()	Odpytuje urządzenie o wybrane elementy tj. nazwa, producent etc. i porównuje je z tymi zdefiniowanymi w konstruktorze, jeżeli coś się nie zgadza zwraca błąd
connect()	Metoda łączy się z urządzeniem (gdy metoda check() nie zgłosi błędu)
wr(„komenda")	Przesyła komendę na urządzenie
query(„komenda?")	Przesyła komendę na urządzenie, żądając odpowiedzi
rd(liczba)	Odczytanie danej liczby bajtów z bufora urządzenia

Tablica 1.3. *Metody z klasy GpibDev ułatwiające tworzenie oprogramowania dla urządzeń GP-IB.*

Istnieje jeszcze kilka innych metod pomocniczych, ale ich funkcjonalność jest bardzo zbliżona do wspomnianych wyżej (czasami istniała potrzeba nieznacznego przebudowania owych metod, aby dostosować ich działanie do starszych urządzeń, na przykład urządzenia firmy Hewllet-Packard z lat 70-tych zamiast obecnie standardowej komendy **IDN?* (komenda zwraca informacje o urządzeniu) rozpoznawały komendę *ID* lub *IDN*), z racji tego metody te nie są one tutaj przytoczone.

Sterownik urządzenia jest klasą o nazwie zgodnej z danym urządzeniem(na przykład dla urządzenia HP6624a, klasa nazywa się hp6624a). Każda taka klasa dziedziczy po klasie *Gpibdev* wspomniane wyżej metody. Aby lepiej zobrazować sposób tworzenia sterownika, poniżej przedstawione zostaną części kodu wraz z opisem przykładowego programu sterującego. Jest to oprogramowanie dla zasilacza firmy Hewllet-Packard oznaczonego jako HP6624A. Każdy sterownik posiada dwa pliki, nagłówkowy o nazwie *nazwaurządzenia.h* i drugi z ciałem metod *nazwaurządzenia.cpp.* W przytaczanym przypadku są to pliki *hp6624a.cpp* i *hp6624a.h.*

Przykładowy plik *hp6624.h*:

```
...
#include "../utils/stringutils.h"
#include "gpibdev.h"
...

class hp6624:public GpibDev
{
   public:
   hp6624(bool i_debug=false);
   bool setVoltage( int channel, float voltage);
   ...
};
```

[2-3]: Oprócz klasycznych plików nagłówkowych dołączane są dodatkowe. Biblioteka stringutils zawiera metody pomocnicze do obsługi zmiennych typu sringów (na przykład obcinanie, scalanie, zamian zmiennej typu int na string etc.), jej opis znajduje się w pliku naglówkowym. Plik nagłówkowy gpibdev, zawiera wspomnianą klasę z metodami pomocniczymi do komunikowania się z urządzeniami

[6-12]: Ciało klasy w którym umieszczane są deklaracje metod takich jak widoczna metoda setVoltage(int, float). Jak widać klasa dziedziczy po klasie GpibDev. Dodatkowo klasa posiada jawny konstruktor. Budowa plików nagłówkowych z reguły nie jest skomplikowana i posiada jedynie wskazane wyżej elementy.

Kolejnym plikiem jest plik *hp6624.cpp*, zawierający definicje zadeklarowanych metod. Oto ważniejsze elementy wnętrza tego pliku.

```
...
#include "hp6624.h"
...
hp6624::hp6624(bool i_debug)
{
 m_BoardIndex=0;
 m_ManufactorName="Hewlett Packard";
 m_DeviceName="HP6624A;
 m_Device=31256;
 m_PrimaryAddress = 5;
 m_SecondaryAddress = 0;
 debug = i_debug;
}
...
bool hp6624::setVoltage( int chn, float volt )
```

```
16  {
17    string cmd = "VSET "+StringUtils::IntToStr(chn)+","+
18
19    wr(cmd);
20    string c;
21    c = query("VSET? "+StringUtils::IntToStr(chn));
22    cout << "Ustawiono: " << c  << "V  na kanale " << chn << ".\n";
23    return true;
24    }
25  ...
```

[2]: Na początku należy dołączyć informacje o pliku nagłówkowym z deklaracją metod.

[4-13]: Konstruktor w którym umieszczono informacje o urządzeniu (są one następnie sprawdzane przez metodę check() z klasy GpibDev). Urządzenie jest odpytywane o dany element, jeżeli istnieje rozbieżność pomiędzy konstruktorem, a odpowiedzią urządzenia metoda nie pozwala na połączenie.

[15-24]: Definicja przykładowej metody. Tworzona jest zmienna typu string z komendą, która przesłana ma być do urządzenia. W tym wypadku jest to komenda VSET. Posiada ona dwa parametry to jest: kanał (zmienna chn) na którym napięcie/prąd ma zostać ustawione (zasilacz ten posiada cztery kanały wyjściowe), i wartość owego napięcia/prądu (zmienna volt, dla większości urządzeń opis komend znajduje się w podręczniku programistycznym). Gdy komenda zostanie stworzona, następuje przekazanie jej do urządzenia za pomocą metody wr(komenda). Kolejne elementy zawarte w metodzie są elementami pomocniczymi. Otóż tworzona jest kolejna komenda, następnie przesyłana jest ona za pomocą metody query(komenda) do urządzenia. Powoduje to iż urządzenie zwraca do programu odpowiedź w formie zmiennej typu string. W przytoczonym przykładzie służy to sprawdzeniu, czy odpowiednie napięcie faktycznie zostało poprawnie ustawione. Wynik w tym wypadku wyświetlany jest w terminalu za pomocą standardowego wyjścia.

Podsumowując, typowy sterownik składa się z dwóch plików. W pliku nagłówkowym deklarowane są odpowiednie metody i tworzone jest ciało klasy, która dziedziczy po klasie *GpibDev*, co zapewnia dostęp do metod ułatwiających sterowanie urządzeniami. W drugim pliku definiowane są zadeklarowane metody. To w tym pliku umieszcza się odwołania do odpowiednich komend, definiuje się obsługę odebranych danych, wykonuje obliczenia etc.

Tworząc wspomniane pliki użytkownik finalnie posiada klasę, którą następnie użyć może w celu stworzenia właściwego programu sterującego urządzeniem. W laboratorium do tego celu stosowano darmową bibliotekę *TCLAP*[3]. Jest to biblioteka która w łatwy sposób umożliwia tworzenie programów konsolowych ze zdefiniowanymi parametrami wejściowymi. W dalszej części przedstawiony zostanie sposób wykorzystania owej biblioteki, do stworzenia kompletnego programu wykorzystującego zbudowaną klasę dla urządzenia (sterownik). Program wynikowy jest programem konsolowym, co stanowiło przyjęte założenie, aby mógł on być wykorzystywany do budowy systemów pomiarowych w sposób jaki zażyczy sobie tego użytkownik (czyli albo oparty o skrypty, albo o środowisko graficzne).

Przedstawiony plik do przytoczonej wyżej klasy, to także plik dla zasilacza HP6624A. Nie ustalono (tak jak w przypadku sterownika) sposobu nazywania tych plików. Autor przyjął

konwencję nazewniczą tych plików jako *nazwaurządzenia-dev.cpp*. Oto najważniejsze elementy pliku *hp6624-dev.cpp*.

```
...
#include "tclap/CmdLine.h"
#include "hp6624.h"'
...
int main(int argc,char *argv[])
{
  #ifdef __WIN32__
  if (!LoadDll())
  {
    printf ("Unable to correctly access the 32-bit GPIB DLL.\n");
    return 1;
  }
  #endif

  try
  {
    CmdLine cmd("HP6624 example", ' ', "0.001 alfa");
    ValueArg<std::string> setVolt("v", "set_volt",
          "Sets voltage on HP6624", false, "string");
    cmd.add( setVolt);
    cmd.parse( argc, argv );
    if(setVolt.isSet())
    {
      hp6624 hp6624;
      hp6624.connect();
      hp6624.setVoltage(1, setVolt.getValue());
    }
  }
  catch (ArgException &e)
  {
    cerr << "error: " << e.error() << " for arg " << e.argId()
                                          << endl;
  }
  #ifdef __WIN32__
    FreeDll();
  #endif
```

```
37      return 0;
38  }
```

[2] Oprócz typowych plików nagłówkowych, dołączono dwa dodatkowe. Pierwszy to plik nagłówkowy z metodami biblioteki *TCLAP*. Drugi plik, to plik nagłówkowy wskazujący dla jakiego urządzenia tworzony będzie program konsolowy.

[5] Całość umieszcza się w metodzie *main()*. Oczywiście istnieje możliwość zdefiniowania dodatkowych metod wcześniej i następnie użycie ich wewnątrz metody *main()* celem wykonania dodatkowych obliczeń etc.

[7-13] Kawałek kodu pomiędzy słowami *ifdef*, a *endif* umieszcza się aby dołączyć plik biblioteki obsługującej standard GP-IB w środowisku Windows.

[17] Stworzenie nagłówka do tworzonego programu konsolowego.

[18-19] Następnie definiowane są odpowiednie parametry zgodnie z tym jak wymaga tego biblioteka *TCLAP*. Istnieje kilka typów parametrów poprzez które wywoływana może być metoda. Tutaj użyto typu *ValueArg*, który oznacza tyle, iż zdefiniowany parametr przyjmuje jedną wartość wejściową (Wspomnieć należy, iż parametry mogą być multi-argumentowe, co pozwala na przekazywanie do metod więcej niż jednego parametru, przy jednym wywołaniu programu. Uzyskuje się to poprzez typ *MultiArg*, opis wszystkich typów dostępny jest w dokumentacji biblioteki TCLAP). Typ danych wejściowych ustalono na string. Dalej podawana jest nazwa parametru. Wewnątrz nawiasów (po nazwie parametru) zdefiniowana jest nazwa argumentu po użyciu którego dany parametr zostanie wywołany (tutaj *-l*). Kolejne elementy składowe parametru to ekwiwalentna nazwa której użyć można zamiast podania argumentu (tutaj *–set-volt*). Kolejno przedstawiona jest informacja, która pojawia się gdy użytkownik zażąda pomocy (pomoc uruchamia się automatycznie, gdy użytkownik uruchomi program z błędnymi parametrami). W kolejnej linii umieszczono *cmd.add(nazwaparametru)*, które dodaje parametr do listy parametrów.

[20] Następnie wszystkie zdefiniowane parametry są parsowane.

[20-27] Tutaj umieszczono zadania jakie wykonuje wywołanie danego parametru. Jest tutaj odwołanie do metody z klasy urządzenia. W tym przypadku odwołano się do metody *setVolatge(int,float)*. Przyjmuje ona dwa parametry, pierwszy to kanał na zasilaczu na jakim ustawione ma być napięcie, a drugi to wartość napięcia. Port ustawiony jest „na sztywno” w kodzie źródłowym, natomiast wartość napięcia przekazywana jest poprzez argument wywołania. Przed wspomnianą wyżej metodą następuje próba połączenia(metoda *connect()*), jeżeli występują jakieś trudności (błędny adres, niezgodność danych z konstruktora etc.), to zostanie zwrócony błąd i nie zostanie ustawione napięcie.

[29] Na końcu, gdyby coś poszło nie tak, łapane są wszystkie wyjątki.

[34-36] Dodatkowo jeżeli użytkownik pracuje w środowisku Windows, należy zwolnić pamięć po bilbiotece DLL.

Po skompilowaniu, użytkownik otrzymuje konsolowy program wykonywalny. Program ten można wywołać jako: *./nazwaprogramu -l wartość-argumentu*, co spowoduje wywołanie metod umieszczonych wewnątrz parametru. Do owych metod przekazane zostaną wartości argumentów wejściowych.

W taki sposób tworzone były wszystkie sterowniki. Każdy posiada swój konsolowy program uruchamiany z odpowiednimi parametrami. Użytkownik może łączyć takie programy w większe systemy pomiarowe w dwojaki sposób. Na przykład poprzez zbudowanie skryptu systemowego. Możliwe jest także proste linkowanie takich programów do zbudowanego środowiska GUI, gdyż każdy taki program reaguje na pewne parametry wejściowe, a wartości zwracane

są albo do pliku, albo na konsole (co umożliwia łatwe ich przechwytywanie). Oczywiście zaznaczyć należy, iż to budowa sterownika narzuciła taką a nie inna formę środowiska GUI, gdyż najpierw opracowany został sposób tworzenia sterowników, a dopiero potem opracowano koncepcję środowiska graficznego do obsługi urządzeń. Z zalet takiej budowy sterowników, można wymienić bardzo duża swobodę w ich projektowaniu i w obróbce danych odebranych od urządzenia. Użytkownik nie jest także związany z żadnym konkretnym typem kontrolera, gdyż biblioteka na której opiera się oprogramowanie jest darmowa i wspiera większość dostępnych na rynku kontrolerów.

1.3. Urządzenia dla których stworzone zostało oprogramowanie

Stworzono szereg sterowników dla urządzeń pomiarowych, które wykorzystywane były/są w pomiarach wskazanych układów. Wszystkie urządzenia dla których stworzone zostały programy sterujące zaprezentowano poniżej, z opisem przykładowych parametrów wywołania. Część urządzeń była wcześniej oprogramowana przez innych użytkowników, autor jednakże wykorzystywał to oprogramowanie celem budowania wskazanych systemów pomiarowych.

Ogólna uwaga: Jeżeli w kolumnie drugiej występuje argument pisany małymi literami, lub jako cyfra, znaczy to, iż jest to jego domyślna wartość, którą należy wprowadzić do programu. Natomiast jeżeli w kolumnie drugiej występuje więcej niż jeden argument, należy wprowadzać je w następujący sposób *nazwaprogramu -nazwaparametru arg1 -nazwaparametru arg2 ....*

1.3.1. Analizator widma HP4195A

Jest to analizator zaprojektowany na początku lat 80-tych przez firmę Hewllet-Packard. Z racji tego, iż analiza sygnału dokonywana jest w sposób analogowy (poprzez wzmacniacze logarytmiczne), proces analizy jest stosunkowo wolny. Dodatkowo z racji zastosowania wspomnianych wzmacniaczy wymagana jest korekcja sygnału o około 2 dB, co jeszcze mocniej wpływa na szybkość pomiaru. Sam analizator pozwala zbierać widmo w przedziale 1Hz-500MHz, w trybach takich jak SPECTRUM (badanie szumów) i NETWORK (badanie widma częstotliwościowego). Więcej informacji o urządzeniu znajduje się w podręcznikach dla urządzenia [4][5].

Program sterujący miał pozwalać na ustawienie parametrów pomiaru, takich jak szerokość pasma, typ pomiaru, odczytywanie informacji. Dodatkowo oprogramowanie miało z zebranego widma szumowego liczyć numerycznie całkę, celem wyznaczenia wartości widmowej mocy szumów. Poniżej w tabeli 1.4 umieszczono wszystkie parametry dla przykładowego programu *4195a-dev* stworzonego w labolatorium. Programem tym badane były układy elektroniki Front-End, celem określenia szumów jakie w nich występują.

parametr	argument/-y	opis
-a	NUMBER	Ustawia rodzaj analizy. NUMBER=1 to tryb NETWORK (pomiary widma), NUMBER=2 to tryb SPECTRUM (pomiary szumów)
-l	FILENAME	ładuje plik konfiguracyjny. FILENAME=nazwa pliku konfiguracyjnego. Domyślnie jest to 4195a.cnf
-p	NUMBER	wykonuje pomiar. NUMBER=liczba powtórzeń pomiaru, celem określenia statystyki

Tablica 1.4. *Paremtry i argumenty dla analizatora HP4195A.*

Parametry takie jak szerokość pasma, tłumienia, numer portu ustawia się w pliku konfiguracyjnym.

1.3.2. Analizator widma HP4395A

Jest to młodszy brat analizatora HP4195A. Analiza sygnału przebiega już w sposób cyfrowy, co bardzo mocno przyspieszyło proces pomiaru. Tak jak starsza wersja i tutaj użytkownik ma możliwość pomiaru w trybach SPECTRUM i NETWORK. Niestety usunięta została możliwość wyświetlania wyników ze skalami logarytmicznymi. Powiększono natomiast bufor danych do 800 komórek pamięci (starsza wersja HP4195A posiadała ich 400). więcej informacji o urządzeniu znajduje się w podręczniku [6].

Program sterujący miał umożliwiać zbieranie widma, a także jako dodatkową opcje, liczenie z zebranego widma całki numerycznej metodą trapezów, celem określenia widmowej mocy szumów. Jednakże z racji niemożności przeprowadzania pomiaru w skalach logarytmicznych algorytm do liczenia całki w szerszym paśmie jest bardziej skomplikowany. Dokładny opis rozwiązania tego problemu znajduje się w części pracy dotyczącej pomiarów, dokładnie w opisie statycznych pomiarów układu DAC. W tabeli 1.5 przedstawiono parametry dla programu sterującego *hp4395a-dev*.

parametr	argument/-y	opis
-a	NUMBER	Ustawia rodzaj analizy. NUMBER=1 to tryb NETWORK (pomiary widma), NUMBER=2 to tryb SPECTRUM (pomiary szumów)
-l	FILENAME	ładuje plik konfiguracyjny. FILENAME=nazwa pliku konfiguracyjnego. Domyślnie jest to 4195a.cnf
-c	NUMBER	ustawia częstotliwość środkową. NUMBER=częstotliwość środkowa w [Hz]
-s	NUMBER	NUMBER=szerokość pasma w [Hz]
-p	NUMBER	wykonuje pomiar. W trybie SPECTRUM liczy całke z widma, poprzez co określony zostaje poziom szumów. iczba powtórzeń pomiaru, celem określenia statystyki

Tablica 1.5. *Paremtry i argumenty dla analizatora HP4395A.*

1.3.3. Analizator urządzeń półprzewodnikowych B1500A

Jest to bardzo precyzyjne urządzenie, które umożliwia nastawianie i odczytywanie prądów i napięć z bardzo dużą dokładnością. Urządzenie jest konfigurowalne, możliwe jest zamontowanie maksymalnie sześciu jednostek SMU (z ang. Single Measurement Unit). Jednostki te różnią się funkcjonalnością. Dla przykładu istnieją moduły SMU pozwalające na ustawianie wyższych napięć i prądów (HPSMU, z ang. *High Power SMU*), inne pozwalają na wykonywanie pomiarów z dokładnością 10 pA (HRSMU, z ang. *High Resolution SMU*) (istnieje możliwość zwiększenia dokładności do 1pA stosując moduł ASU, z ang. *Atto Sense Unit*). Urządzenie na którym pracował autor wyposażone było w pięć modułów SMU, dwóch zwykłym, jednego HPSMU i jednego HRSMU. Ostatnim modułem jest moduł dostarczający masę. Więcej informacji o urządzeniu znajduje się w podręcznikach [7][8].

Program miał pozwalać na mierzenie i ustawianie napięć i prądów. Dodatkowo celem wykonania pomiarów statycznych zaimplementowano obsługę zewnętrznego wyzwalacza(triggera). Oczywiście istnieją też funkcje do pobierania danych z urządzenia. W tabeli 1.6 zaprezentowano parametry wywołania dla programu *b1500a-dev.*

parametr	argument/-y	opis
-v	NRSMU, RANGE, VOLTAGE, ICOMPLIANCE, 0, 0	Ustawia napięcie na wybranym SMU. Wartości RANGE podaje się zgodnie z podręcznikiem użytkownika(domyślnie ustawić można 0, co oznacza AUTO). Wartości VOLTAGE i ICOMPLIANCE podaje się w Voltach i Amperach, są to wartość napięcia i ograniczenie prądowe. Dwa ostatnie argumenty są domyślne, jeżeli jednak użytkownik chciałby je zmienić należy sięgnąć do podręcznika [8].
-i	NRSMU, RANGE, CURRENT, VCOMPLIANCE,0,0	analogicznie jak z napięcie, z tym, iż na wybranym SMU ustawiany jest prąd
-f	reset	resetuje urządzenie. Należy zwrócić uwagę, iż po wystąpieniu poważniejszej awarii(na przykład rozłączeniu przewodów podczas pracy w trybie wyzwalacza), najlepszym sposobem na zresetowanie urządzenia jest reset całego systemu Windows zainstalowanego na urzadzeniu
-o	open	otwiera dany SMU
-z	close	zamyka dany SMU
-d	NUMBER	ustawia format danych gromadzonych w buforze urządzenia. NUMBER = 0 oznacza typ domyślny, czyli z nagłówkiem. Dane „surowe”, czyli bez nagłówka to NUMBER=12. Inne formaty dostępne w podręczniku [8]
-m	TYPE, NRSMU, NUMBER	ustawia rodzaj pomiaru na wybranym SMU. Argument TYPE może przyjmować wartości 1 i 2, które oznaczają kolejno pomiar pojedyńczy i pomiar typu SWEEP. Argument NUMBER to określenie, czy użytkownik mierzyć chce napięcie(NUMBER=1), czy prąd(NUMBER=2)
-x	start	uruchamia skonfigurowany pomiar. Ręczny wyzwalacz

parametr	argument/-y	opis
-g	NUMBER	pobiera dane z bufora urządzenia. Argument NUMBER oznacza liczbę bajtów jaka może zostać przesłana w jednym cyklu. Optymalna wartość to NUMBER=100(taka też ustawiana jest domyęlnie)
-e	1	ustawia urządzenie w tryb zewnętrznego TRIGGERA (wyzwalacza)
-t	1,1,-1,-1	ustawia opcje dla zewnętrznego TRIGGERA. Dokładny opis dostepny w podręczniku przy komendzie TGP
-p	NUMBER	wykonuje pomiar. W trybie SPECTRUM liczy całke z widma, poprzez co określony zostaje poziom szumów. iczba powtórzeń pomiaru, celem określenia statystyki

Tablica 1.6. *Paremtry i argumenty dla analizatora urządzeń półprzewodnikowych B1500A.*

1.3.4. Miernik wartości LCR, HP4284A

Urządzenie pomiarowe do mierzenia wielkości takich jak C,R,G,L etc. Miernik testuje szukane wielkości podając sygnał o przebiegu przemiennym z określoną przez użytkownika amplitudą i częstotliwością. Możliwa jest zewnętrzna polaryzacja za pomocą dostarczonych narzędzi, i zewnętrznego zasilacza. Urządzenie wykorzystywane było w pomiarach pojemności przy badaniu krzemowych sensorów japońskiej firmy Hammamatsu. Więcej informacji o urządzeniu w podręczniku [9].

Zadaniem programu sterującego miało być ustawienie parametrów pomiaru (czyli częstotliwości i amplitudy sygnału testowego, oraz innych parametrów potrzebnych do prawidłowego pomiaru), następnie wykonanie pomiaru i odczytanie/zapisanie danych. W tabeli 1.7 zaprezentowano parametry i argumenty dla programu *hp4284a-dev*.

parametr	argument/-y	opis
-a	NUMBER	Ustawia amplitudę dla sygnału testowego w [V]. NUMBER=amplituda
-f	NUMBER	ustawia częstotliwość sygnału próbkującego w HZ/KHZ/MHZ. Zapis 10000 i 10KHZ są sobie tożsame i oba poprawnie rozpoznawane przez program
-c	TYPE, STATE	konfiguruje pomiar. Argument TYPE to rodzaj pomiaru, czyli TYPE = „CPRP"/"CSRS" (reszta możliwych argumentów dostępna w podręczniku użytkownika). Argument STATE przyjmuje dwie wartości, STATE="ON" włącza zewnętrzną polaryzację, STATE="OFF" wyłącza
-r	start	uruchamia skonfigurowany pomiar
-s	short	wykonuje korekcje przy zwartych przewodach pomiarowych
-o	open	korekcja przy rozwartych przewodach pomiarowych
-g	data	pobiera dane z urządzenia i zapisuje do pliku

Tablica 1.7. *Paremetry i argumenty dla miernika LCR HP4284A.*

1.3.5. Analizator urządzeń półprzewodnikowych HP4145B

To starszy analizator urządzeń półprzewodnikowych (odpowiednikiem jego jest opisany już B1500A). Z racji wieku (urządzenie pochodzi z początku lat 70 XX wieku) nie obsługuje standardu GP-IB w najnowszej wersji (czyli wspierającej język SCPI), a już dość archaiczną wersję HP-IB. Nastręczało to trochę problemów, gdyż wymagało przerobienia metod pomocniczych z klasy *GpibDev*. Urządzenie nie rozpoznaje, obecnie uważanej za standardową, komendy *IDN?. Zamiast niej potrzeba było stworzyć metodę wykorzystującą komendę ID. Dodatkowo urządzenie nie wspiera instrukcji do zwracania danych, dlatego też informacje musiały być przesyłano po bajcie i „ręcznie" łączone w całość. Kolejnym utrudnieniem jest całkowity brak kontroli błędów podczas transmisji. Na urządzenie wysłać można praktycznie dowolny rozkaz, a urządzenie nie zwraca błędu. Należało więc kontrolować transmisję rozkazów poprzez oprogramowanie.Więcej informacji o urządzeniu znajduje się w podręczniku [10]

Program sterujący miał obsługiwać podstawowe funkcje takie jak nastawianie i odczytywanie napięcia z wybranej jednostki SMU (urządzenie ma na stałe wbudowane 4 jednostki SMU). Miał służyć jako „wersja zapasowa" dla analizatora B1500A. W tabeli 1.8 parametry dla programu sterującego *4145b-dev*.

parametr	argument/-y	opis
-u	start	Przełącza urządzenie w tryb USER(użytkownika)
-v	CHANNEL, RANGE,VOLTAGE, ICOMPLIANCE	nastawia napięcie na wybranym SMU z odpowiednimi parametrami(znaczenie podobne jak dla analizatora B1500A)
-i	CHANNEL, RANGE, CURRENT, VCOMPLIANCE	to samo tylko ustawiany jest prąd

Tablica 1.8. *Paremetry i argumenty dla analizatora urządzeń półprzewodnikowych HP4145B.*

1.3.6. Zasilacz HP6624A

To prosty zasilacz, posiadający cztery wyjścia, które służyć mogą jako źródła napięciowe, lub prądowe. Posiada podstawowe funkcje takie jak ustawianie ograniczenia prądowego i napięciowego, czy funkcję pozwalającą na pomiar napięcia i prądu (niestety z dość słabą w porównaniu z innymi urządzeniami rozdzielczością sięgającą 2 mV). Więcej informacji o urządzeniu znajduje się w podręczniku [11]

Program miał dostarczać parametrów pozwalających na proste nastawianie napięcia i prądu. Dodatkowo umożliwiono nastawianie ograniczenia prądowego i napięciowego, jak także dodano możliwość pomiaru wskazanych wielkości. W tabeli 1.9 parametry dla programu *hp6624a-dev*.

parametr	argument/-y	opis
-v	VOLATGE	Ustawia napicie na wybranym wyjściu
-i	CURRENT	ustawia prąd na wybranym wyjściu
-p	get	mierzy prąd na wybranym wyjściu
-n	get	mierzy napięcie na wybranym wyjściu
-l	FILENAME	ładuje plik konfiguracyjny. Domyślnie 6624a.cnf

Tablica 1.9. *Paremetry i argumenty dla zasilacza HP6624A.*

Parametry takie jak numer wyjścia, czy ograniczenia prądowe i napięciowe ustawia się w pliku konfiguracyjnym (domyślnie nazwa tego pliku to 6624a.cnf).

1.3.7. Generator SMB100

Jest to dokładny generator przebiegów prostokątnych i sinusoidalnych. Posiada mody pracy pozwalające na ustawienie częstotliwości pracy w zakresie 0.1Hz - 3GHz. Generator ten posiada bardzo dobry stosunek sygnału do szumów, co udało się potwierdzić podczas pomiarów opisanych w rozdziale 6.3. Dodatkowe informacje o urządzeniu dostępne są w podręczniku użytkownika [12].

Celem programu miało być ustawianie odpowiedniego przebiegu, z określonymi parametrami (amplituda, częstotliwość etc.). W tabeli 1.10 przedstawiono parametry i argumenty dla programu *smb100-dev*.

parametr	argument/-y	opis
-f	NUMBER	Ustawia częstotliwość sygnału
-a	NUMBER	ustawia amplitudę sygnału
-p	NAME	ustawia rodzaj przebiegu (na przykład NAME="SINE" ustawia przebieg sinusoidalny)
-x	start	uruchamia przebieg. Ręczny wyzwalacz

Tablica 1.10. *Paremetry i argumenty dla generatora SMB100.*

1.3.8. Generator AWG2021

Urządzenie to nie zostało oprogramowane w pełni przez autora. Dodał on tylko kilka niezbędnych funkcji. Było one jednak wykorzystywane i w pomiarach statycznych układu DAC, jak również w analizie jakościowej sygnału dostępnych generatorów. Dlatego też przytoczone zostaną parametry z jakimi mogą być uruchamiane programy sterujące. Generator pracować może z zakresem częstotliwości do 2.5MHz. Zegar pracować może do 250MHz. Więcej informacji o urządzeniu w podręcznikach [13][14]. W tabeli 1.11 przedstawiono parametry i argumenty dla programu *awg2021-dev*.

parametr	argument/-y	opis
-f	NUMBER	Ustawia częstotliwość sygnału
-a	NUMBER	ustawia amplitudę sygnału
-l	FILENAME	ładuje plik konfiguracyjny. Znajdują się w nim wszystkie neizbędne parametry. Jest to najbardziej wskazany sposób konfiguracji urządzenia. Nazwa domyślna dla pliku konfiguracyjnego to awg.cnf
-r	FILENAME	ładuje przebieg zdefiniowany przez użytkownika (należy wcześniej odpowiednio przygotować pliki z przebiegiem). Sposób tworzenia takiego pliku opisay jest w podręczniku[14]

Tablica 1.11. *Paremetry i argumenty dla generatora AWG2021.*

1.3.9. Generator AFG3102

To młodsza wersja generatora AWG. Posiada szersze pasmo sięgające 100MHz. Jak sie jednak okazało podczas pomiarów jest urzadzeniem mniej dokładnym niż generator AWG. Więcej

informacji o urzadzeniu dostępnych jest w podręczniku [15]

W tabeli 1.12 przedstawiono parametry dla programu sterującego *afg3102*.

parametr	argument/-y	opis
-s	SHAPE, FREQ, AMP, OFF, POLAR, STATE	Ustawia rodzaj przebiegu. SHAPE to ksztal, czyli SIN, PULS, RAMP. FREQ to częstotliwość. AMP to amplituda impulsu. POLAR to polarność(przyjmuje wartości true i false). Natomiast STATE to włączenie lub wyłączenie przebiegu
-r	run	wyzwala przebieg na urzadzeniu. Ręczny wyzwalacz
-l	FILENAME	ładuje plik konfiguracyjny. Znajdują się w nim wszystkie neizbędne parametry. Jest to najbardziej wskazany sposób konfiguracji urządzenia. Nazwa domyślna dla pliku konfiguracyjnego to afg.cnf

Tablica 1.12. *Paremetry i argumenty dla generatora AFG3102.*

1.3.10. Oscyloskopy serii TDS

Jest to seria urządzenia z tej samej rodziny, funkcje jakie posiadają są bardzo zbliżone. Umożliwiło to napisanie wspólnego oprogramowania sterującego. Obsługa urządzeń jest szybka i wygodna, nie sprawiają one żadnych problemów podczas pisania oprogramowania. Wykorzystywane funckje są typowimi jakich można by się spodziewać po oscyloskopach. Konfiguracja przebiega głównie poprzez plik konfiguracyjny. Ważniejsze parametry można także ustawić „ręcznie".Więcej informacji znajduje się w podręczniku [16]

Ważniejsze funkcje jakie obsługuje oprogramowanie przedstawiono w tabeli 1.13.

parametr	argument/-y	opis
-r	STATE	Uruchamia/kończy pomiar. STATE przyjmuje wartości RUN/STOP
-c	CHANNEL, STATE	włącza lub wyłącza dany kanał oscyloskopu. STATE przyjmuje argument OPEN/CLOS, natomiast CHANNEL to numer kanału od 1 do 4
-t	CHANNEL, TYPE	typ pomiaru na danym kanale.TYPE to rodzaj pomiaru, czyli AC/DC/GND
-l	FILENAME	ładuje plik konfiguracyjny. Domyślnie jest to tds.cnf. Znajdują się w nim wszystkie inne nieprzytoczone tutaj opcje jakie mogą zostać nastawione na wspomniane urządzenia. Preferowany sposób nastawienia parametrów

Tablica 1.13. *Paremetry i argumenty dla oscyloskopów TDS firmy TEKTRONIX.*

Rozdział 2

Przegląd koncepcji dotyczących projektowanego środowiska GUI

Znając już sposób tworzenia programów sterujących dla urządzeń pomiarowych, w rozdziale tym opisane zostaną koncepcje jakie narodziły się co do projektowanego środowiska GUI. Sama potrzeba stworzenia takiego środowiska zrodziła się dużo później, niż sposób tworzenia sterowników, dlatego też GUI musiało być zaprojektowane pod potrzeby sterowników, a nie odwrotnie. Sam pomysł stworzenia środowiska graficznego narodził się, aby ułatwić użytkownikom możliwość konfiguracji wskazanych urządzeń. Główne założenia dla projektowanego środowiska to elastyczność, rozumiana jako łatwa i dowolna rozbudowa systemu o nowe elementy. A także pełna kompatybilność z napisanymi w przeszłości programami sterującymi (na przykład tymi dla generatorów). Założenia te nakładają szereg ograniczeń co do wybranych technologii. Podczas prac nad środowiskiem rozważano wiele narzędzi, które mogły by zostać użyte do budowy systemu graficznego. W rozdziale tym zostaną one przedstawione, z argumentacją, która zadecydowała o ich porzuceniu lub akceptacji.

2.1. Środowisko graficzne na bazie biblioteki GTK 3.0 języka C++

Biblioteka GTK 3.0 [17] jest biblioteką do tworzenia graficznych interfejsów użytkownika przy wykorzystaniu GTK. Dedykowana jest dla języka programowania C++. Wspiera obsługę systemów operacyjnych m. in. Linux i Windows.

Koncepcja zakładała wykonanie nakładki graficznej, do której podpinane były by już binarne wersje oprogramowania sterującego urządzeniami. Wykonane zostały aplikacje testowe, których zadaniem miała być odpowiedź na pytanie, czy możliwe jest zapewnienie odpowiedniej elastyczności oprogramowania? Pierwsze testy wskazały na duże problemy z implementacją nowych elementów. Każdorazowo wymagało to dopisania do programu dodatkowego kodu i rekompilacji całego środowiska graficznego. Konieczność rekompilacji całości jest bardzo dotkliwa, szczególnie przy środowisku bardziej rozbudowanym (a zakładać należy taki właśnie, najgorszy z punktu widzenia analizy zalet i wad, przypadek). Możliwe było by tworzenie interfejsu w oparciu o pluginy, ale to rozwiązanie także nie zapewnia wystarczającej elastyczności. Dodatkowo patrząc na bibliotekę GTK pod kątem zestandaryzowania, i niezmienności jej funkcji w czasie, zauważono, iż ryzyko wystąpienia zmian w kolejnych wersjach biblioteki (a co się z tym wiąże w projektowanym środowisku) jest większe niż na przykład w oprogramowaniu pisanym w oparciu o język Java. Te dwa główne argumenty stanowiły o porzuceniu możliwości wykorzystania biblioteki GTK jako narzędzia do budowy interfejsu. Jak zostało wspomniane wykonano jedynie proste próbne aplikacje, z bardzo ograniczonymi możliwościami.

Dodać tutaj należy, iż przy wyborze technologii kierowano się odczuciami subiektywnymi o danym narzędziu i miano na względzie wszystkie ograniczające czynniki, takie jak na przykład ograniczona ilość czasu, czy praca nad projektem w pojedynkę. Patrząc na możliwość wykorzys-

tania GTK pod szerszym kątem, przy odpowiednio dużym zaangażowaniu możliwym było by stworzenie środowiska elastycznego i spełniającego stawiane wymagania (na przykład poprzez wspomniane pluginy). Jednakże wymagało by to zbyt dużego nakładu pracy względem dostępnego na ukończenie projektu czasu, co z racji ograniczonych możliwości było nie do zaakceptowania (szczególnie, iż praca nie miała dotyczyć samego środowiska graficznego, a całości rozumianej jako tworzenie sterowników i wykonywanie próbnych pomiarów). Nie brano pod uwagę bibliotek takich jak na przykład biblioteka Qt [18], ponieważ związane są z nią pewne ograniczenia licencyjne, a także podobne ograniczenia jak te wynikające z GTK 3.0. Dla przykładu wspomniana biblioteka Qt oparta jest na zmodyfikowanej licencji GPL [19], co skutkuje tym, iż aby być zgodnym z duchem tej licencji, należy korzystać jedynie z oprogramowania wolnego opartego o tylko tą licencję.

Podsumowując, po analizie możliwości ukończenia projektu w terminie, okazało się iż stworzenie GUI w oparciu o GTK, przy zachowaniu wszystkich założeń, było by zbyt trudne. Dlatego też zaczęto poszukiwania innego narzędzia.

2.2. Środowisko graficzne oparte o technologie internetowe

Kolejny pomysł opierał się na stworzeniu serwisu internetowego w językach skryptowych na przykład PHP/Ruby [20], za pomocą którego zlecane były by odpowiednie modele pomiarowe. Następnie aplikacje serwerowe sterowały by urządzeniami i zwracały odpowiedź do użytkownika. Języki skryptowe takie jak Ruby, są językami które w prosty sposób pozwalają na budowę nawet skomplikowanych serwisów internetowych. Dodatkowo Ruby posiada dość potężne narzędzie zwane Rails, które jeszcze bardziej ułatwia proces tworzenia takich serwisów. Rozważając tę koncepcję miano na uwadze także fakt bardzo dużego zestandaryzowania wszystkich technologi internetowych, co zapewnia niezmienność wykorzystywanych technologii w czasie i pełną kompatybilność z kolejnymi wersjami.

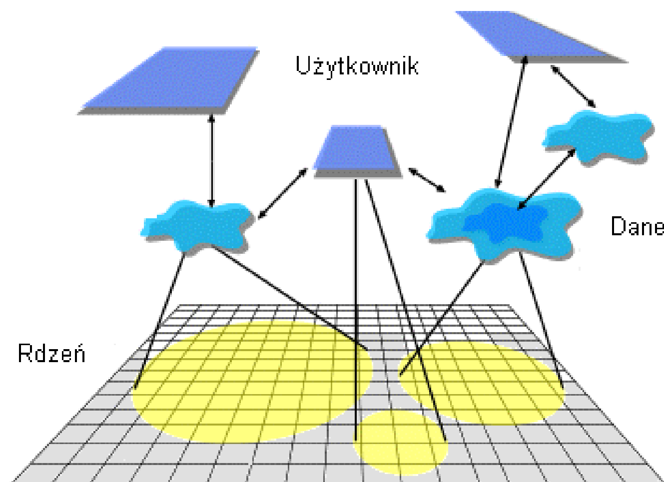
Projekt ten nie doczekał się jednakże realizacji. Analizując zalety i wady autor doszedł do wniosku, iż trudno było by zapewnić odpowiednią swobodę w dodawaniu nowych elementów do środowiska. Wymagało by to dostępu do serwera przez każdego użytkownika, co mogło by sprawiać problemy z bezpieczeństwem, jak także z synchronizacją wykonywanych zadań. Dodatkowo istniał problem uruchomienia samego serwera i odpowieniedo skonfigurowania, co także zajęzło by znaczną ilość czasu. Dlatego też koncepcja ta nie była wdrażana. Przytoczona ona została jako jedna z hipotetycznych możliwości, które były brane pod uwagę, jednakże ze względu na występujące według autora trudności, nie doczekała się realizacji. O jej odrzuceniu zadecydował także fakt porównania jej z technologiami opisanymi w kolejnym podrozdziale.

2.3. Środowisko tworzone przy wykorzystaniu koncepcji programowania z użyciem aktorów

Jest to koncepcja, która zakładała wykorzystanie narzędzia bardziej złożonego, które w łatwy sposób pozwoli budować modele pomiarowe. Wybrano dwa narzędzia, to znaczy Dive jako środowisko graficzne, a także Ptolemeusza jako kompletne rozwiązanie dostarczające wszystkie niezbędne składniki potrzebne do stworzenia interfejsu graficznego sterującego pracą wybranych urządzeń.

2.3.1. Środowisko graficzne Diva

Jest narzędziem, które służy do budowania intuicyjnych interfejsów użytkownika [21]. Środowisko napisane zostało w Javie, a główne jego założenia to jak największa elastyczność i prostota w implementacji elementów. Środowisko graficzne zbudowane w oparciu o Dive, wykorzystuje programowanie zorientowane aktorowo. Użytkownik w głównym polu programu z angielskiego nazwanym „workflow”, buduje z gotowych elementów w formie klocków pewną hierarchię pomiędzy nimi (łącząc elementy za pomocą „sznurka” metodą „chwyć i przeciągnij” znaną chociażby z języka G i środowiska LabView). Każdy element posiada zdefiniowaną liczbę portów wejściowych i wyjściowych, które użytkownik może łatwo łączyć. Każdy port reprezentuje dane określonego typu, dlatego też przesłanie informacji na wybrany port wymusza pewną akcję. Uproszczona struktura środowiska zaprezentowana została na rysunku 3.1.



Rysunek 2.1. *Architektura środowiska Diva. Istnieje struktura warstwowa, na samym dole zbudowany jest rdzeń na którym opiera się oprogramowanie. Dalej istnieje warstwa danych, które są obrabiane. Na samej górze znajduje się użytkownik, który poprzez interfejs steruje całością.*

Na podstawie Divy stworzonych zostało wiele narzędzi (takich jak na przykład Kepler, czy Mescal). Jest to narzędzie w pełni darmowe, oparte na liberalnej licencji, której treść zawarta jest w załączniku C.

Pierwotnie chciano stworzyć od podstaw własne środowisko graficzne opierając się na Divie, jednakże po pewnym czasie okazało się, iż wymagało by to zbyt dużo czasu. Dlatego też zdecydowano się na poszukiwania narzędzia, które także korzysta z Divy, a dostarcza już pełnego zestawu narzędzi. Wybór padł na Ptolemeusza.

2.3.2. Środowisko pomiarowe Ptolemeusz

To narzędzie, podobnie jak Diva, stworzone na Uniwersytecie w Berkeley [22], wykorzystuje Dive jako część interfejsu graficznego. Przeznaczeniem narzędzia są symulacje dla rożnych dziedzin nauki, wykonywane w czasie rzeczywistym, lub jako oprogramowanie współbieżne. Wykorzystuje się je do obliczeń prowadzonych dla Biologii, Chemii, Geografii etc. W oparciu o Dive stworzone zostało środowisko graficzne nazwane Vergil, które jest już kompletnym interfejsem zaadaptowanym do potrzeb środowiska. To właśnie narzędzie uznano za idealne i użyto jako podstawę do stworzenia GUI sterującego urządzeniami pomiarowymi w oparciu o

oprogramowanie sterujące napisane w C++.

Sam Ptolemeusz jest narzędziem bardzo rozbudowanym. Rozwijany jest od lat 80-tych XX wieku. W środowisku tworzy się modele, których podstawowym elementem są aktorzy. Wybrano to narzędzie ponieważ dostarcza wszystkich niezbędnych elementów aby w łatwy sposób zaimplementować do niego stworzone sterowniki, posiada praktycznie gotowe GUI i wygodną obsługę danych. Ptolemeusz tak jak Diva napisany został w Javie i pracuje na Wirtualnej Maszynie Java (JVM). Dzięki temu jest to narzędzie niezależne od systemu operacyjnego. Wykorzystanie Javy do budowy środowiska powoduje, iż oprogramowanie stwarza możliwości łatwej przebudowy i rozszerzania o nowe komponenty. Dodatkowo w Ptolemeuszu zastosowano język XML celem określania aspektów konfiguracyjnych środowiska. To znaczy istnieje rdzeń programu, który o wszystkich pakietach dowiaduje się poprzez odpowiednie pliki XML. Chcąc więc dodać nowy element nie ma potrzeby przebudowywać całości. Dodaje się po prostu nowy plik XML, lub odpowiednią informację w pliku już istniejącym, i po odpaleniu środowiska, zmiana zostanie automatycznie zauważona. Jest to ta cecha, której brak było bibliotece GTK 3.0 i próbom tworzenia GUI przy pomocy języka C++. Tutaj każda zmiana wprowadzona może być bardzo szybko i prosto. Nie bez znaczenia, podczas wyboru tego narzędzia, był też fakt iż jest ono rozwijane przez bardzo długi czas i dobrze sprawdzone pod kontem wykorzystanych technologii. A na jego podstawie powstało już bardzo wiele innych programów. Wspomnieć można chociażby o programie Kepler, który ma umożliwiać podobne symulacje jak Ptolemeusz, ale z wykorzystaniem sieci gridowych.

Nie skorzystano oczywiście z całego środowiska Ptolemeusz. Wykorzystany został jedynie główny rdzeń oprogramowania, głównie środowisko graficzne Vergil [23] (które jest, jak wspomniano, kompletnym środowiskiem graficznym dostarczającym także narzędzi do obsługi danych). Na tej podstawie zbudowane zostało oprogramowanie sterujące urządzeniami pomiarowymi. Inne elementy, takie jak aktorzy, dyrektorzy, dodatkowe komponenty, nie zostały włączone do projektu. Zostały one zastąpione nowymi, napisanymi pod stawiane wymogi.

2.4. Podsumowanie rozważanych koncepcji

Wszystko co zostało przedstawione w powyższym rozdziale, z oczywistych względów obarczone jest subiektywnymi ocenami wybranych technologii. Dlatego też wybór padł na to, a nie inne narzędzie. Starano się pokazać, iż wybór którego dokonano jest faktycznie najefektywniejszy. Oczywiście, nie jest wykluczone, iż da się stworzyć podobne środowisko korzystając z odrzuconej przez autora biblioteki GTK, lub przy wykorzystaniu języka skryptowego Ruby, czy Python. Ale jak było już wspomniane, wybrane narzędzie, pozwala w szybki sposób stworzyć środowisko na tyle kompletne, iż możliwym stało się wykonywanie testowych pomiarów. Na potwierdzenie w rozdziale szóstym zaprezentowano szereg pomiarów, które wykonane zostały już za pomocą tego środowiska.

Rozdział 3

Aktor jako pojęcie w programowaniu

Ponieważ pojęcie aktora w pracy występuje bardzo często, w rozdziale tym przedstawiona zostanie historia powstania koncepcji opartej na programowaniu opartym o aktorów, jak również jej ewolucja na przestrzeni lat. Na pierwszy rzut oka, pojęcie to jest zbliżone do koncepcji obiektu, zostaną jednak przedstawione główne różnice, a także zalety jakie płyną przy korzystaniu z aktorów w opozycji do programowania obiektowego(OOP). Rozdział podzielony jest na dwie części, w pierwszej zaprezentowano historię, i dwa różniące się podejścia do owej koncepcji. Natomiast w części drugiej przedstawiono analizę korzyści jakie daje wykorzystanie aktorów.

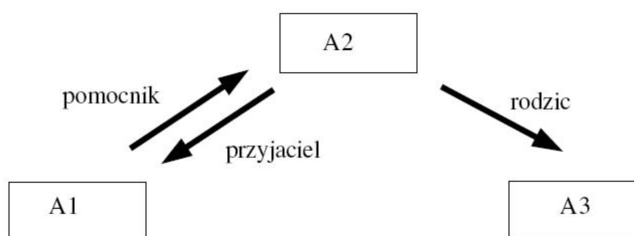
3.1. Prace Carla Hewitta i Gul Agha

Po raz pierwszy pojęcia aktor w ujęciu programowania użył Carl Hewitt z *Artifical Inteligence Laboratory* przy MIT (*Massachusetts Institute of Technology*). Praca opublikowana została w roku 1977, a dotyczyła symulacji zachowań ludzkich [24].

Według Hewitta aktor miał być bardzo ogólnym agentem obliczeniowym, posiadającym w swej strukturze:

1. Zbiór funkcji (zachowań) na informacje płynące z zewnątrz
2. Zbiór danych, które przesyłane są do innych aktorów na skutek interakcji z otoczeniem
3. Mechanizm powoływania do życia kolejnych aktorów

Bardzo ważną cechą aktora zaproponowanego w owej pracy była asymetryczność działań, co lepiej zobrazować można na prostym schemacie, przedstawionym na rysunku 3.1.



Rysunek 3.1. *Asymetryczność działań aktorów*

Na przytoczonym schemacie:

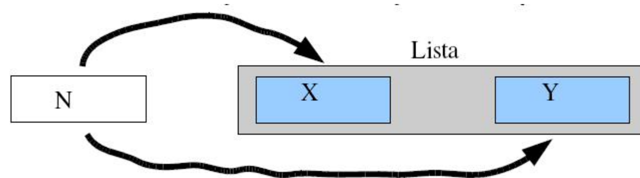
— Aktor A1 wie, iż aktor A2 jest jego „pomocnikiem”
— Aktor A2 wie, iż aktor A1 jest jego „przyjacielem”
— Aktor A2 wie, iż aktor A3 to jego „rodzic”

A więc występuje pewna asymetria w działaniu pomiędzy aktorami A2 i A3 (A3 nie posiada informacji o A2, czyli nie może wpływać na jego zachowania, natomiast A2 może wymuszać

określone zachowania na A3). Ogólnie zbiór takich zależności pomiędzy aktorami, został przez Hewitta nazwany „zbiorem zależności” i stanowi, jak było to wspomniane, jedną z dwóch głównych cech opisu aktorów.

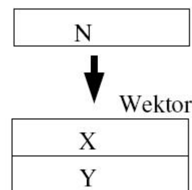
Co ważne, aktor definiowany jest poprzez zbiór zachowań (w pewien sposób wirtualnych), a nie poprzez realizacje tych zachowań. Jest to bardzo ważne założenie, bo stawia pojęcie aktora na poziomie o znacznej ogólności.
Dla zobrazowania można sobie wyobrazić aktora, który ma zwrócić na określone działanie liczbę w postaci dwóch innych liczb. Pierwsza, X określa część całkowitą, a druga Y resztę. Aktor niech nazwany będzie N.
Z tego co zostało napisane wyżej, wyniak iż istnieje pewna mnogość realizacji zachowań takiego aktora. Dwie przykładowe przytoczone zostały na rysunkach 3.2 i 3.3.

1. Aktor N zwraca wartość jako listę zawierającą elementy X i Y



Rysunek 3.2. *Przykład realizacji poprzez listę*

2. Aktor zwraca wartość jako wektor ze współrzędnymi X i Y



Rysunek 3.3. *Przykład realizacji poprzez wektor*

Wynika stąd wspomniany już bardzo istotny fakt, iż aktor definiowany jest poprzez odpowiednie zachowania (w tym przypadku zwracanie wartości całkowitej X i reszty Y), a nie ich realizacje (których może być więcej niż jedna!).
Wyjaśnia to także termin aktor, którego użył Hewitt. Taki model może posiadać kilka realizacji, czyli może wcielić się w kilka ról (ale za każdym razem jest to ta sama osoba).

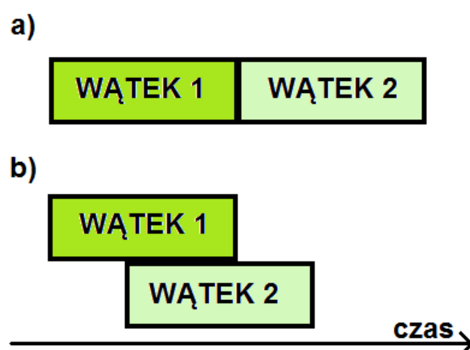
W późniejszym okresie nad zagadnieniem aktorów pracował współpracownik Hewitta, Gul Agha (także pracujący na MIT). Rozwijał on pomysł programowania aktorowo zorientowanego w aspekcie wykorzystania w programowaniu współbieżnym. Opublikował on szereg prac ([25], [26], [27]) w których zajmuje się problemem dostosowania koncepcji aktorów do wymagań współbieżności. Aktor z założenia posiada pewne cechy, które wskazują, iż mógłby być idealnym narzędziem w programowaniu równoległym. Na przykład posiada określoną jedynie funkcjonalność, a nie gotowy zbiór danych (zmiennych), co nie powoduje trudności ze spełnieniem warunku dzielenia zasobów (jeden z podstawowych problemów w programowaniu współbieżnym). Agha wspólnie z Hewittem pokazali [28], iż zastosowanie aktorów umożliwia bardzo łatwy i szybki sposób na tworzenie systemów współbieżnych. Zaproponowali nowy język programowania nazwany ACT3, który pracował na wielu maszynach połączonych w sieć. Język

stworzony został w oparciu o LISP, tworzony był zbiór aktorów, a następnie przesyłane były pomiędzy nimi komunikaty. Całość kontrolowana była za pomocą systemu skryptów. Dzięki temu narzędziu pokazano, iż języki bazujące na aktorach posiadają cechy, które umożliwiają zapewnienie dużej równoległości przy zachowaniu cech dziedziczenia. Nie ma problemów z podziałem zasobów, a cały system bardzo łatwo można dynamicznie przekonfigurować.

Podsumowując zaprezentowano podstawowe cechy programowania bazującego na aktorach. Obecnie bardzo dużo wyspecjalizowanych narzędzi korzysta z tej filozofii. Wspomnieć można o środowisku *LabView*, narzędziach Simulink, czy programach CADowskich do składania tekstu w drukarni. Przy stworzeniu odpowiedniego interfejsu graficznego i kontroli przepływu informacji, programowanie orientowane na aktorów daje bardzo dużą elastyczność i łatwość w konstruowaniu modeli. Dlatego też staje się coraz bardziej popularne. Oczywiście wszystkie te narzędzia są jedynie pewną realizacją oryginalnej filozofii zaproponowanej przez Hewitta. Część nie spełnia niektórych postulatów, ale sama filozofia programowania aktorowo orientowanego jest w tych narzędziach dobrze widoczna.

3.1.1. Programowanie współbieżne

W miejscu tym należy zrobić krótką dygresje, aby lepiej zobrazować czym jest programowanie współbieżne, o którym było wspomniane przy pracach Gula Agha i Carla Hewitta. W programowaniu sekwencyjnym wszystkie operacje ustawiane są w przysłowiową „kolejkę”, i wykonywane jest jedynie jedno zadanie, w jednym cyklu pracy zegara procesora. Cały czas istnieje jeden wątek, który tożsamy jest najczęściej z jednym procesem. W programowaniu współbieżnym wątków może istnieć wiele i wszystkie wykonywane mogą być w tym samym czasie (lub tym samym przedziale czasu). Ważnym jednakże jest aby podkreślić, iż zgodnie z najpopularniejszą definicją, dana liczba (grupa) wątków działa pod kontrolą jednego procesu. Oczywiście istnieć może kilka procesów, ale każdy posiada zespół własnych wątkó.Różnice między wykonywaniem wątków sekwencyjnie i współbierznie pokazano na rysunku 3.4.



Rysunek 3.4. *a) programowanie sekwencyjne b) współbieżne(wielowątkowe).*

Nasuwa się pytanie, jak zrealizować współbieżność, skoro wiadomo, iż procesory są obiektami sekwencyjnymi? Sposobów jest kilka. Najbardziej oczywisty, to procesory wielordzeniowe. Zapewnia to rzeczywistą pracę programu jako współbieżnego. W idealnym przypadku każdemu rdzeniowi (z ang. *core*, stąd nazwy nowych procesorów *Dual Core* etc.) jednostki CPU przypisujemy jeden wątek co w pełni spełnia założenia programowania współbieżnego.

Inne rozwiązanie to podzielenie mocy obliczeniowej procesora jednordzeniowego (lub ogólnie zasobów, wliczając pamięć etc.) na części, i każda taka część przeznaczona jest dla danego

wątku. Nazywa się to dzieleniem zasobów. Rozwiązanie to jest dość wygodne, gdyż umożliwia tworzenie praktycznie dowolnej ilości wątków i nie wymaga wielordzeniowych jednostek CPU. Ograniczeniem jednak jest moc procesora, gdyż im więcej wątków, tym mniej każdy z nich ma mocy i zasobów obliczeniowych.

Jeszcze innym sposobem osiągnięcia wielowątkowości jest wykorzystanie do obliczeń kilku (lub kilku tysięcy) maszyn. Jako przykład można podać wspomnianą kilka razy w pracy sieć gridową. Są to sieci bardzo rozległe, w których pracują nawet dziesiątki tysięcy procesorów (połączone najczęściej w klastry obliczeniowe, takie jak na przykład klaster w krakowskim CYFRONET). Na dobrą sprawę dopiero w takich sieciach objawia się cała moc programowania współbieżnego. Istnienie takiej ilości procesorów i pamięci pozwala na bardzo duże usprawnienie skomplikowanych obliczeń. W pracy wspomniane już było narzędzie Kepler[29], które jest rozwinięciem narzędzia Ptolemeusz właśnie na możliwość obliczeń w sieciach gridowych. Oczywiście Kepler także bazuje na programowaniu orientowanym na aktorów.

3.2. Różnice i podobieństwa pomiędzy obiektem, a aktorem

Na zakończenie rozdziału warto zadać sobie pytanie, czym aktor różni się od obiektu? Co po części można już wnioskować z opisu koncepcji aktorów przedstawionej powyżej, główna różnica to dużo większa ogólność aktora względem obiektu. Z założenia aktor jest agentem obliczeniowym, posiada szereg funkcji, które są niezależne od realizacji. Natomiast obiekt, pracuje na danych.

Najważniejszą jednak różnicą jest fakt, iż za pomocą aktorów da się zbudować każdy system oparty o obiekty, a w drugą stronę to nie działa (zbyt mała ogolność obiektów). Jak było wspomniane jedną z cech aktora jest to, iż może on powoływać do życia kolejnych aktorów. Własności tej nie ma programowanie obiektowe (obiekt może co najwyżej „budzić” inny obiekt), dlatego też nie da się za pomocą programów obiektowych stworzyć w pełni funkcjonalności programowania orientowanego na aktorów.

Różnice widać także na przykładzie środowiska Ptolemeusz. W środowisku tym aktor definiowany jest jako zbiór metod i właśnie obiektów, czyli obiekt może być elementem składowym aktora (ale nie musi). Podsumowując, aktor jest narzędziem dużo bardziej ogólnym i daje większe możliwości jeżeli chodzi o kwestie obliczeń, czy programowania współbieżnego. Środowisko odpowiednio zaprojektowane daje naprawdę duże możliwości i ogromną elastyczność tworzonego oprogramowania.

Rozdział 4

Środowisko GUI

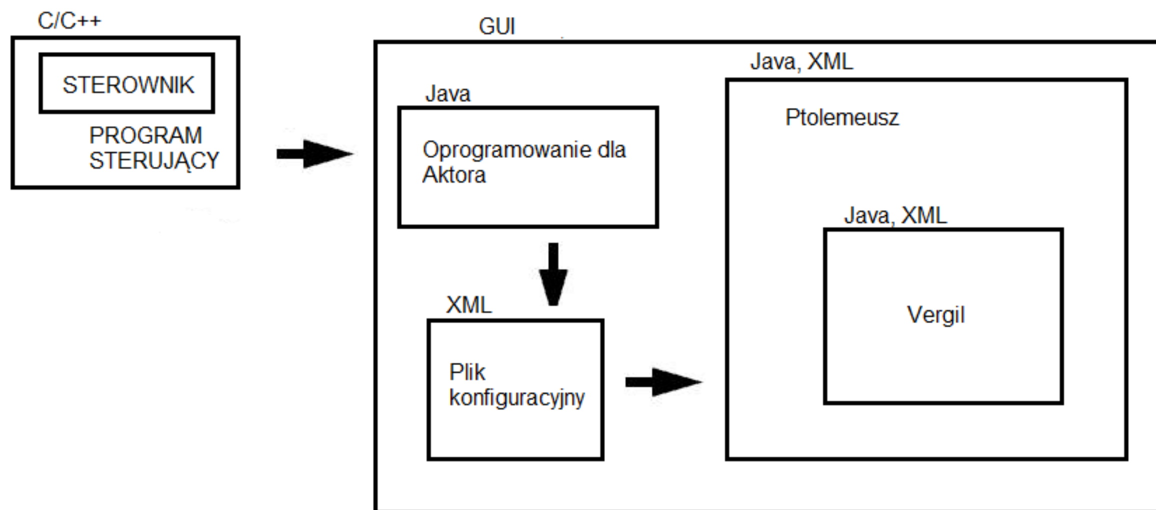
Rozdział ten opisuje środowisko graficzne stworzone do sterowania urządzeniami pomiarowymi. Kilkakrotnie było już wspomniane, iż środowisko graficzne GUI charakteryzować musi się dużą elastycznością. Podczas prac nad oprogramowaniem, w labortorium pojawiło się kilka nowych urządzeń, które musiały zostać dodane do środowiska. Ponieważ sposób tworzenia sterowników dla urządzeń był już bardzo dobrze określony, to środowisko GUI musiało być pod niego dostosowane, a nie odwrotnie. Kilkakrotnie także pojawiała się konieczność stworzenia nowych aktorów o zadaniach czysto informatycznych, jak na przykład zapis do pliku, czy odpowiednia obróbka stringów. Pokazuje to, iż środowisko musi być łatwe w re-konfiguracji i rozszerzaniu o nowe elementy. Dodatkowo Ptolemeusz (czyli narzędzie na którym opiera się stworzone GUI) zawierał mnóstwo elementów, które na potrzeby autora nie były użyteczne. Należało je albo usunąć, albo przebudować. W tym rozdziale autor postara się przybliżyć budowę GUI, przedstawi także aspekty, które świadczą o dużej elastyczności środowiska.

4.1. Ogólna struktura środowiska graficznego

Całość środowiska graficznego oparta jest na wspomnianym już narzędziu Ptolemeusz. Środowisko to zbudowane jest za pomocą języka Java, dodatkowo wykorzystywany jest także język XML do budowy plików konfiguracyjnych. Założono, iż każdy aktor w środowisku graficznym reprezentować ma, albo daną funkcję urządzenia, albo (gdy urządzenie ma prostszą funkcjonalność) całe urządzenie. Ogólną strukturę zaproponowanego środowiska przedstawiono na rysunku 4.1.

Założono, iż każdy aktor ma być oddzielnym wątkiem systemowym, odpowiedzialnym za wybraną funkcję urządzenia. Co więcej, w idealnym przypadku, powinna istnieć korelacja jedna funkcja urządzenia - jeden aktor. Często z racji mocno określonej funkcjonalności urządzeń aktor reprezentował całe urządzenie. Dopiero z takich bardzo wyspecjalizowanych aktorów powinna istnieć możliwość budowania systemów bardziej ogólnych, z określoną przez użytkownika funkcjonalnością. Dzięki odziedziczonym po Ptolemeuszu narzędziom takie rozwiązanie okazało się bardzo wydajne i nie skomplikowane pod względem implementacji. Java dostarcza łatwej możliwości tworzenia wątków, jak również sprawowania nad nimi kontroli (przysyłanie i odbieranie inforamcji od wykonywanego sterownika).

Każdy aktor obsługujący urządzenia pomiarowe, to uruchomiony program sterujący dla urządzenia z wywołanymi odpowiednimi argumentami. Dla przykładu, użytkownik posiada program sterujący do woltomierza-zasilacza, nazwany „zasilacz”. Jest to typowy sterownik stworzony zgodnie z regułami podanymi w rozdziale dotyczącym sterowników. Niech sterownik ten posiada szereg parametrów i argumentów z jakimi może być wywołany. przykładowo „-v”, „-i”, służa do ustawiania napięcia i prądu etc. Wobec wspomnianych wyżej założeń, każdemu z tych parametrów w środowisku graficznym odpowiadać powinien oddzielny aktor. Po umieszczeniu w



Rysunek 4.1. *Struktura budowanych systemów pomiarowych uwzględniająca środowisko graficzne.*

środowisku danego aktora, gdy użytkownik uruchomi model, tworzony jest wątek z odpalonym w nim programem sterującym z danym parametrem. Taka struktura umożliwia budowanie bardziej złożonych modeli z już gotowych elementów. Programy w formie binarnej sterujące urządzeniami, znajdują się w katalogu *cbin/* w głównym katalogu środowiska graficznego. Na obecnym etapie wymagana jest manualna aktualizacja tego oprogramowania. To znaczy, jeżeli w programie sterującym nastąpi jakaś zmiana, użytkownik powinien sam przegrać zaktualizowany program do wspomnianego katalogu, celem zastąpienia starszej wersji. W przyszłości planowana jest automatyzacja tego procesu, poprzez ustalanie w środowisku miejsca gdzie znajdują się programy w formie binarnej, co umożliwiło by stworzenie modułu do automatycznej aktualizacji (na przykład wykorzystując SVN).

4.1.1. Pliki konfiguracyjne.

Struktura środowiska odziedziczona została po Ptolemeuszu. Czyli tak samo jak w nim istnieje szereg plików konfiguracyjnych pisanych w języku XML. Z tych plików informacje pobiera środowisko, i na tej podstawie ustalane są parametry uruchomieniowe GUI. Pliki konfiguracyjne dostępne są w katalogu *configs/*. Te najważniejsze przytoczone zostały w tabeli 4.1.

nazwa pliku	opis
basiActorLibrary.xml	informacje o aktorach
basicDirectors.xml	informacje o reżyserach
defaultFullConfiguration.xml	ustawienia dla środowiska, strona tytułowa, rozdzielczość etc.

Tablica 4.1. *Najważniejsze pliki konfiguracyjne.*

Pliki konfiguracyjne ładowane są automatycznie podczas uruchamiania programu. Umożliwia to bardzo proste dodawanie nowych elementów, czy edycję już istniejących. Dla przykładu podana zostanie zawartość pliku *basicActorLibrary.xml*, aby zobrazować istotę działania tych plików.

```
/**** NAGŁÓWEK *****/
<?xml version="1.0" standalone="no"?>
<!DOCTYPE entity PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">

/***** ODNOŚNIKI*****/
<group>
  <input source="ptolemy/actor/lib/other.xml"/>
  <input source="ptolemy/actor/lib/devices.xml"/>
  <input source="ptolemy/actor/lib/io/io.xml" />
</group>
```

Jak widać z przytoczonego przykładu, zamiast odnośnika do kodu źródłowego w pliku istnieje powiązanie do kolejnych plików XML (za pomocą tagu *input*). Tutaj są to pliki *other.xml*,*devices.xml* i *io.xml*. Taka budowa pozwala na tworzenie hierarchii w menu gdzie użytkownik wybierać będzie aktorów (środowisko tworzy tą hierarchię automatycznie, właśnie na podstawie plików konfiguracyjnych). Poniżej przedstawiono zawartość pliku *io.xml*.

```
// [1]
<?xml version="1.0" standalone="no"?>
<!DOCTYPE plot PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">

// [2]
<entity name="IO" class="ptolemy.moml.EntityLibrary">

// [3]
<configure>
  <?moml
  <group>
  <doc>Actors for reading and writing into files.</doc>

  <entity name="LineReader"
                         class="ptolemy.actor.lib.io.LineReader">
  <doc>Read one line at a time from a text file
                       and output as a string</doc>
  </entity>

  <entity name="LineWriter"
```

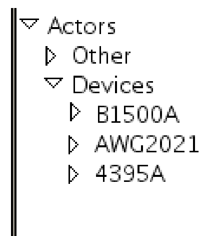
```
22                              class="ptolemy.actor.lib.io.LineWriter">
23          <doc>Write the value of a string token,
24                                  one per line, to a text file.</doc>
25     </entity>
26      </group>
27     ?>
28  </configure>
29
30  </entity>
```

[1]: Jak w każdym pliku konfiguracyjnym, istnieć musi identyczny nagłówek, po którym środowisko rozpoznaje pliki konfiguracyjne.

[2]: Tag *name* służy do nadawania nazwy, która wyświetlana będzie w środowisku graficznym. Tag *class* mówi środowisku, iż ma stworzyć nową bibliotekę z aktorami.

[3]: To tutaj umieszcza się wszystkie informacje o aktorach, które zawierać ma biblioteka. Na początku w tagach *doc* umieszcza się dokumentację dla danej biblioteki widoczną w środowisku. Następnie definiuje się informacje na temat aktora, w tagu *name* nazwę aktora, w tagu *class* nazwę klasy z odpowiedniego pliku Java, gdzie znajdują się definicje funkcji aktora.

Po umieszczeniu wszystkich informacji we wspomnianych plikach i po uruchomieniu środowiska w menu wyboru aktorów powstanie hierarchia jak na rysunku 4.2.



Rysunek 4.2. *Przykładowa hierarchia stworzona poprzez pliki konfiguracyjne XML.*

W dalszej części pracy wyjaśnione zostanie, gdzie dokładnie umiejscowione jest w interfejsie okno wyboru aktorów. Tutaj zostało to przytoczone celem zaprezentowania możliwości konfigurowania środowiska za pomocą plików XML.

4.2. Informacje na temat kodu źródłowego, najważniejsze klasy, metody i typy

W podrozdziale tym omówione zostały aspekty techniczne dotyczące kodu źródłowego jaki użytkownik może wykorzystać w celu budowy nowych elementów.

4.2.1. Parametry, Port-Parametry i Porty

Podstawowym narzędziem, które użytkownik dziedziczy po środowisku graficznym, są definicje portów i parametrów. Jak wiadomo każdy aktor może posiadać ich założoną przez użytkownika

ilość. W środowisku są one jednak zróżnicowane. Najprostszym typem jest parametr. Znajduje się on wewnątrz aktora i jego zawartość nie może byc przesyłana na zewnątrz wprost, możliwe jest to tylko przy wykorzystaniu portu. Parametr definiuje się w kodzie jako:

```
value = new Parameter (this, "'Name"');
```

Domyślnie typem jego jest STRING, aby zmienić typ, po zadeklarowaniu dodać trzeba:

```
value.setTypeEquals(BaseType.DOUBLE);
```

Oprócz typów takich jak wspomniane STRING i DOUBLE, rozpoznawane są jeszcze INT i BOOLEAN. Inną ważną funkcją jaką wykonać można na parametrach jest nadanie im domyślnej wartości. Robi się to poprzez metodę:

```
value.setExpression(DOMYSLNAWARTOSC);
```

Oprócz parametrów użytkownik definiować może też tak zwane port-parametry. Posiadają one identyczne funkcje jak parametry, a definiuje się je jako:

```
value = new PortParameter (this, "'Name"');
new Parameter(value.getPort(), "_showName", BooleanToken.TRUE);
```

Dzięki wykorzystaniu port-parametru, na ikonie aktora w modelu pojawia się port o wskazanej nazwie. Za pomocą tego portu przekazywana może być do aktora informacja pochodząca z innych aktorów. Możliwe jest też nastawianie wartości takie jak przy zwykłym parametrze, czyli korzystając z opcji aktora. Za pomocą port parametrów definiuje się porty wejściowe. Aby zdefiniować nowy port wyjściowy skorzystać trzeba z typu *TypedIOPort*.

```
out = new TypedIOPort(this, "'out"', false, true);
out.setTypeEquals(BaseType.DOUBLE);
```

W ten sposób zdefiniować można port wyjściowy o zadanym typie i nazwie. Następnie z portu tego korzystać można w programie celem na przykład wysłania na niego odpowiednich danych, które przekazywane są dalej innym aktorom.

4.2.2. Metody prefire(), fire(), postfire() i wrapup()

Są to metody wykonywane gdy użytkownik uruchomi model. Różnią się one sposobem wykonania, aby umożliwić na przykład tworzenie iteracji w modelu.

Najważniejszą metodą jest metoda *fire()*. Wywoływana jest ona przy każdym uruchomieniu modelu (także w kolejnych iteracjach). Za każdym razem zmienne w niej zdefiniowane tworzone są na nowo, co dyskwalifikuje tę metodę jeżeli chcemy iterować wartość wybranej zmiennej. Trochę inaczej działa metoda *prefire()*, która wykonywana jest przed uruchomieniem modelu, w celu zdefiniowania odpowiednich zmiennych, dalej działa analogicznie jak metoda *fire()*. Podobnie działa metoda *postfire()*, jednakże wykonuje się dodatkowo po zakończeniu działania modelu. Natomiast metoda *wrapup()* wykonywana jest jedynie raz po zakończeniu działania modelu. Jest ona bardzo przydatna jeżeli chcemy wyczyścić określone zmienne globalne, lub zachować ich wartość na wszystkie iteracje. Aby stworzyć na przykład wspomnianą iterację wybranej wielkości(napięcia etc.), trzeba skorzystać z metod *fire()* i *wrapup()*. W

pierwszej z nich wykonuje się obliczenia, nastawianie parametrów etc. Natomiast w drugiej nadawane są domyślne wartości zmiennym po zakończeniu działania modelu, inaczej każde kolejne uruchomienie modelu prowadziło by do tego, iż zmienne miały by wartości ustalone w poprzednim uruchomieniu modelu. Ilość iteracji ustawia się w reżyserze, w jego pierwszym parametrze. Widać tutaj w praktyce różnice pomiędzy owymi dwoma metodami. Gdyby nie istaniała metoda *wrapup()* iteracja była by niemożliwa, gdyż każde kolejne uruchomienie modelu korzystało by z wartości zapisanych po wcześniejszym uruchomieniu aktora. Metody *prefire()* i *postfire()* mogą być natomiast wykorzystywane w aktorach, gdzie dana wartość ma byc niezależna od iteracji modelu i dany blok funkcji wykonywany ma być zawsze tylko raz, odpowiednio na początku lub na końcu działania modelu.

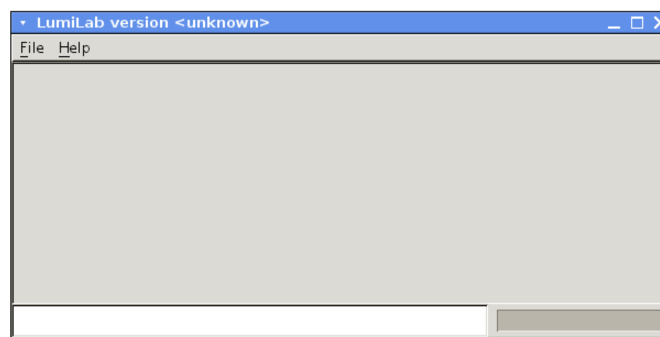
Podkreślić należy, iż to w metodzie typu *fire()* powinny znajdować się wszystkie funkcje jakie reprezentować ma dany aktor. To tutaj znajdują się także wszystkie dane, które potem po wykonaniu przesyłane są na porty wyjściowe. W metodach tych znajdować powinny się też wszystkie obliczenia i obsługa informacji uzyskanych od innych aktorów. Z punktu widzenia sterowania urządzeniem, to tutaj tworzony jest wątek z programem sterującym, a następnie z poziomu tej metody jest on obsługiwany. W dalszej części pracy zaprezentowany jest przykład wykorzystania owej metody do budowy aktora sterującego urządzeniem pomiarowym.

4.3. Główne elementy środowiska graficznego

Opis środowiska podzielony został na podrozdziały w których zawarto najważniejsze moduły GUI. Przedstawiono informacje opisujące najistotniejsze elementy, aby przybliżyć obsługę środowiska.

4.3.1. Uruchamianie i okno powitalne

Środowisko uruchamiane jest poprzez napisany w tym celu skrypt systemowy. Nazywa się on *runLumi*, a znajduje się w głównym katalogu programu. Po uruchomieniu oczom użytkownika ukazuje się okno powitalne programu, przedstawione na rysunku 4.3.

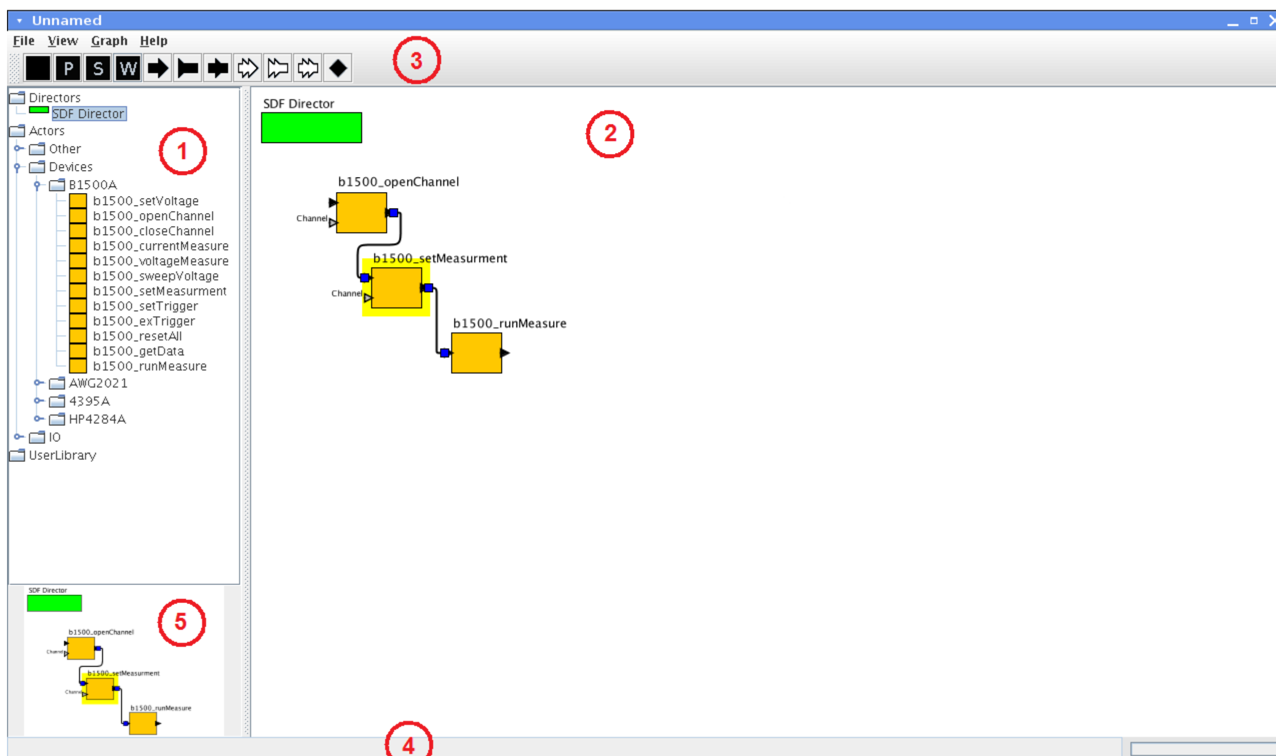


Rysunek 4.3. *Okno powitalne, umieszczone w nim są podstawowe informacje na temat programu.*

Podzielić je można na dwa elementy, czyli menu górne i okno główne. W oknie głównym wyświetlane są informacje podstawowe dotyczące wersji środowiska i nazwy. Natomiast poprzez menu, użytkownik wybiera interesującą go opcję. Poprzez menu „FILE” użytkownik uruchomić może tzw. „GRAPH EDITOR”, czyli edytor graficzny do budowy modeli. Inne opcje są typowymi opcjami użytkowymi tak jak „SAVE”, „OPEN”. Istnieje także możliwość uruchomienia prostego notatnika tekstowego, do zapisywania plików tekstowych z krótkimi notatkami.

4.3.2. Okno tworzenia modelu. Opis elementów interfejsu

Aby stworzyć nowy model wybrać trzeba opcję „GRAPH MODEL” z menu „FILE”. Wtedy oczom użytkownika ukazuje się okno tworzenia modelu widoczne na rysunku 4.4.



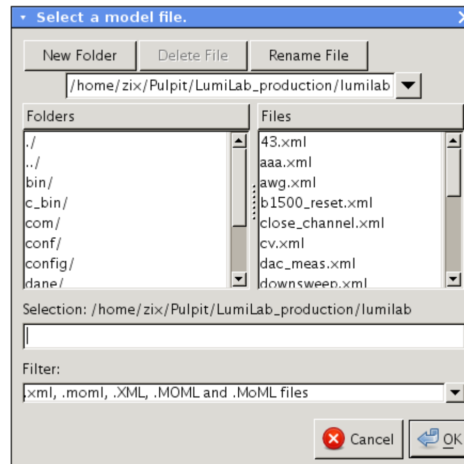
Rysunek 4.4. *Okno tworzenia modelu.*

Okno to podzielić można na pięć części. Jako (1) oznaczono okno gdzie wybiera się elementy składające się na model, czyli reżysera i aktorów. Istnieje menu rozwijane, którego wygląd definiowany jest poprzez pliki konfiguracyjne, a więc jest ono w pełni dostosowywalne do potrzeb użytkownika. Przez (2) oznaczono okno główne, gdzie umieszcza się elementy. To w tym oknie tworzy się model. Dokładny opis tworzenia modelu przedstawiono w jednym z dalszych rozdziałów. Poprzez (3) oznaczono menu z przyciskami akcji. Dostępne są tutaj przyciski podstawowe, służące do sterowania modelem, czyli „PLAY”, „WAIT” i „STOP” (Oznaczone jako standardowe znaki P,S,W). Dalej dostępne są przyciski pozwalające na umieszczenie w modelu portów zewnętrznych. Wykorzystywane są one do tworzenia aktorów pochodnych (o tym także dokładniej napisano w rozdziale dotyczącym tworzenia aktorów). Na górze natomiast znajduje się pasek z menu rozwijalnymi. W menu „FILE” dostępne są podstawowe opcje takie jak otworzenie nowego okna modelu „GRAPH”, zapisanie obecnego modelu „SAVE”, otwarcie zapisanego modelu „OPEN”(rysunki 4.5,4.6). W menu „VIEW” dostępne są opcje do tworzenia aktorów pochodnych, służy do tego opcja „CREATE HIERARCHY”. Przez (4) oznaczono pasek na którym wyświetlane są komendy środowiska. Podstawowe to „executing”, która informuje, iż model się wykonuje. Natomiast przez (5) oznaczono podgląd na całość modelu w miniaturce.

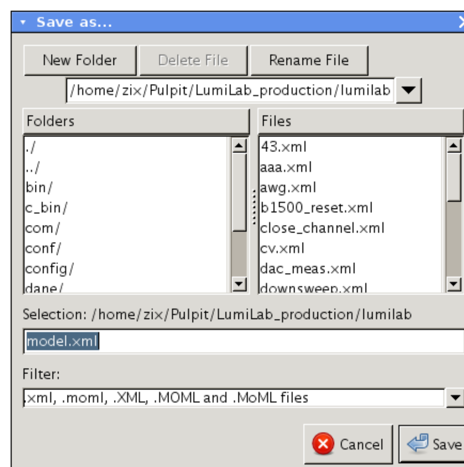
W oknie tworzenia modelu dostępne są skróty klawiszowe(tabela 4.2), które ułatwiają obsługę środowiska.

skrót	akcja
CTRL-S	zapisanie modelu do pliku.
CTRL-O	otwarcie zapisanego modelu
CTRL-A	utworzenie aktora pochodnego. wymaga zaznaczenia aktorów z których składać ma się aktor pochodny
CTRL-H	okno pomocy

Tablica 4.2. *Skróty klawiszowe dostępne w oknie „GRAPH”*



Rysunek 4.5. *Okno zapisywania modelu do pliku.*



Rysunek 4.6. *Okno otwierania modelu z pliku.*

4.4. Tworzenie własnych elementów

Aktora stworzyć można albo zupełnie od podstaw, to znaczy stworzony może być nowy plik Java z obsługą nowych funkcji, albo może być też budowany z gotowych aktorów umieszczonych

w środowisku. Dostajemy wtedy aktorów z funkcjonalnością opartą na innych aktorach. Aby jednak móc stworzyć aktora drugim sposobem, oczywiste jest, iż musi zostać stworzona baza aktorów podstawowych. W rozdziale tym przybliżone zostaną oba sposoby tworzenia nowych aktorów.

4.4.1. Tworzenie aktora poprzez nowy plik Java

W tym przypadku, wszystkie zadania jakie pełni aktor zdefiniowane muszą być w odrębnych plikach Java. Poniżej zaprezentowano strukturę takiego przykładowego pliku, który jest wymagany jako główny łącznik pomiędzy zadaniami pełnionymi przez aktora, a środowiskiem GUI. Istnieje w znacznym stopniu, uogólniony przepis na taki plik, ze zbiorem odpowiednich metod z których skorzystać potrafi środowisko graficzne (o których było w rozdziale dotyczącym kodu źródłowego). Pliki różnią się tylko zawartością metod wykonywalnych (*fire()* etc.).

Aby lepiej zobrazować tworzenie takiego aktora, przedstawione zostaną pliki Java dla jednego z nich. Jest to prosty funkcjonalnie aktor, którego zadaniem jest przemiatanie napięcia w zadanym zakresie i dodatkowo, jeżeli zażyczy sobie tego użytkownik, aktor mierzy prąd i przekazuje jego wartość na port wyjściowy. Oczywiście uznaje się, iż istnieje program sterujący urządzeniem. Najpierw opisany zostanie plik Java, a następnie przedstawiony zostanie plik konfiguracyjny do którego wpisać trzeba informacje o fakcie pojawienie się nowego aktora w środowisku.

```
1  package ptolemy.actor.lib;
2
3  /******************  IMPORTY  ************************
4  * Każdy aktor wymaga pewnych bibliotek które zawierają
5  * odpowiednie elementy.
6  *****************************************************/
7  import ptolemy.data.DoubleToken;
8  import ptolemy.data.BooleanToken;
9  import ptolemy.data.Token;
10 import ptolemy.data.expr.Parameter;
11 import ptolemy.actor.parameters.PortParameter;
12 import ptolemy.actor.TypedIOPort;
13 import ptolemy.data.type.BaseType;
14 import ptolemy.kernel.CompositeEntity;
15 import ptolemy.kernel.util.IllegalActionException;
16 import ptolemy.kernel.util.NameDuplicationException;
17 import ptolemy.kernel.util.Workspace;
18 import java.io.*;
19 /******************  NAZWA KLASY *********************
20 * Nazwa klasy
```

```
21 ********************************************************/
22 public class b1500_sweepVoltage extends Transformer {
23  public b1500_sweepVoltage(CompositeEntity container, String name)
24         throws NameDuplicationException, IllegalActionException {
25  super(container, name);
26
27 /******************  PORTY I PARMETRY  ****************
28 * Definicje portów i parametrów, to tutaj określa się
29 * ich domyślne wartości, typy danych jakie reprezentują
30 * każdy z nich widoczny jest w środowisku        w menu
31 * opcji aktora. Definuję się także, czy ma to być
32 * port wejściowy/wyjściowy.
33 ********************************************************/
34     value = new Parameter (this, "Result");
35     offset = new Parameter (this, "Offset");
36     v = new Parameter (this, "Voltage [V]");
37     vrange = new Parameter (this, "Voltage range");
38     vrange.setExpression("0");
39
40     icompliance = new Parameter (this, "I compliance [A]");
41     icompliance.setExpression("100E-6");
42     icpolarity = new Parameter (this, "I compliance polarity");
43     icpolarity.setExpression("0");
44     icrange = new Parameter (this, "I compliance range ");
45     icrange.setExpression("0");
46
47     channel = new PortParameter (this, "channel");
48     new Parameter(channel.getPort(), "_showName",
49                   BooleanToken.TRUE);
50     out = new TypedIOPort(this, "out", false, true);
51     out.setTypeEquals(BaseType.DOUBLE);
52
53     output.setTypeEquals(BaseType.STRING);
54 /******************  IKONA  *************************
55 * Ikona jaka reprezentować będzie aktora definiowana
56 * jest za pomocą SVG. Ustala się kształ i kolor
57 * figury jaka widoczna jest w polu prostokatnym
58 * reprezentującym aktora
```

```
59 * tutaj mamy prostokąt o kolorze pomarańczowym
60 *******************************************************/
61     _attachText("_iconDescription", "<svg>\n"
62                 + "<rect x=\"-25\" y=\"-20\" " + "width=\"50\"
63                 + height=\"40\" " +
64                 + "style=\"fill:orange\"/>\n"
65                 + "\n"
66                 + "</svg>\n");
67     }
68 
69     //ZMIENNE PUBLICZNE, KAŻDY PORT MUSI BYĆ
70     //TUTAJ ZADEKLAROWANY ABY BYŁ
71     //WIDOCZNY DLA METOD STERUJĄCYCH
72     public Parameter value, offset;
73     public TypedIOPort out;
74     public double d;
75 
76     public float a = 0;
77     public String volts = "0";
78 
79     public Parameter v,vrange,icompliance,icpolarity,icrange;
80     public PortParameter channel;
81 
82 /*******************  KLONOWANIE AKTORÓW  ****************
83 * Kopiowanie aktora, sklonowany jest aktor bez
84 * połączeń. Aby mozna było tworzyć "kopie"'aktorów.
85 *******************************************************/
86     public Object clone(Workspace workspace) throws
87             CloneNotSupportedException {
88         b1500_sweepVoltage newObject = (b1500_sweepVoltage)
89                                 (super.clone(workspace));
90 
91         newObject.output.setTypeAtLeast(newObject.value);
92         return newObject;
93     }
94 
95 /*******************  METODA FIRE  ****************
96 * Metoda fire wykonywana jest po uruchomieniu modelu
```

```
97  * To w niej zawarte są najważniejsze czynniki decydujące
98  * o funkcja aktora. To tutaj powoływany jest wątek
99  * z programem sterującym. Tutaj także wykonywane są
100 * wszystkie obliczenia.
101 ******************************************************/
102     public void fire() throws IllegalActionException {
103         super.fire();
104 
105         channel.update();
106 
107         try
108         {
109          String buff = "";
110          String s;
111 
112          float vol = 0;
113 
114          //NADAWANIE ZMIENNYM WARTOŚCI Z PARAMETRÓW
115 float off = Float.valueOf(offset.getValueAsString()).floatValue();
116 
117          String vstart = v.getValueAsString();
118 
119          vol = Float.valueOf(vstart).floatValue() + a;
120 
121          volts = Float.toString(vol);
122 
123          //ŚCIEŻKA DO PROGRAMU STERUJĄCEGO Z
124          //ODPOWIEDNIMI PARAMETRAMI
125          String path = "c_bin/./B1500 -v "+
126           channel.getValueAsString()+" -v "+
127            vrange.getValueAsString()+
128             " -v "+volts+" -v "+
129              icompliance.getValueAsString()+
130               " -v "+icpolarity.getValueAsString()+
131                " -v "+icrange.getValueAsString();
132 
133          //TWORZENIE WĄTKA Z PROGRAMEM STERUJĄCYM
134          Process rt = Runtime.getRuntime().exec(path);
```

```
135
136         //ODCZYTYWANIE ZAWARTOŚCI KONSOLI SYSTEMOWEJ
137         BufferedReader br = new BufferedReader(new
138                 InputStreamReader(rt.getInputStream()));
139
140         //OCZEKIWANIE NA ŚMIERĆ STWORZONEGO WĄTKA
141         rt.waitFor();
142         a+=off;
143
144             //USUWANIE NIEPOTRZEBNYCH ZNAKÓW
145             //ODCZYTANYCH Z KONSOLI
146             while ((s = br.readLine()) != null)
147              {
148                  for(int i=0;i<s.length();i++)
149                  {
150                      if (s.charAt(i) == '#')
151                      {
152                          buff += '\n';
153                      }else{
154                          buff += s.charAt(i);
155                      }
156                  }
157              }
158
159         br.close();
160
161         //NADANIE PARAMETROWI "VALUE" WARTOŚĆI
162         //ZE ZMIENNEJ "VOLTS", PRZEKAZANIE
163         //TEJ WARTOŚCI POPRZEZ TEN PORT NA ZEWNATRZ
164         value.setExpression(volts);
165         value.setStringMode(true);
166
167         d = Double.valueOf(volts).doubleValue();
168
169        //KONTROLA WYJĄTKÓW
170        } catch(Throwable t)
171              {
172                  value.setExpression("There was some errors
```

```
173                                 in actor: \n\n"+t.getMessage());
174                     value.setStringMode(true);
175                 }
176
177             //PRZEKAZANIE WYBRANYCH PARAMETRÓW NA PORTY WYJŚCIOWE
178             Token ot = new DoubleToken(d);
179             out.send(0, ot);
180             output.send(0, value.getToken());
181         }
182
183 /******************* METODA WRAPUP ******************
184 * Metoda wrapup() wykonywana jest PO zakończeniu
185 * działania modelu. Tutaj wykorzystano ją do czyszczenia
186 * zmiennych aby po iteracji miały domyślną wartość. Lub
187 * ustawiane są domyslne inne wartości zmiennych
188 *****************************************************/
189     public void wrapup() throws IllegalActionException {
190         volts = v.getValueAsString();
191         a = 0;
192         super.wrapup();
193     }
194 }
```

Jest to oczywiście tylko przykład i nie wyczerpuje wszystkich możliwości jakie daje środowisko. Pokazuje on jednak ogólny wygląd pliku aktora. Więcej przykładów dostarczają po prostu pliki aktorów i analiza ich kodu źródłowego. Struktura wszystkich jest bardzo podobna, a różnią się funkcjami jakie aktor ma reprezentować.

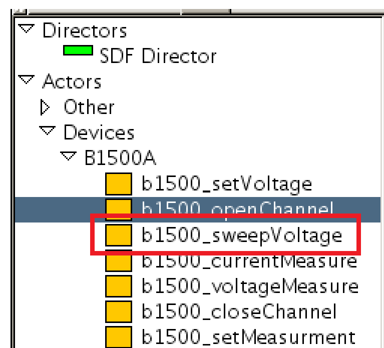
Oprócz pliku Java, aktor musi zostać przypisany do odpowiedniego pliku konfiguracyjnego w którym zawarte są informacje o nazwie aktora, nazwie pliku Java, pomocy etc. Jak było to wspomniane, może to być albo plik *actors.xml*,wtedy aktor pojawi się bezpośrednio w zakładce *Actors*, albo inny plik konfiguracyjny stworzony przez użytkownika i dowiązany do pliku *actors.xml*, wtedy tworzy się odpowiednie podmenu w zakładce *Actors*. Informacje te następnie pobierane są poprzez środowisko automatycznie po uruchomieniu i tworzona jest nowa hierarchia. Oto plik dla opisanego wyżej aktora.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE plot PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">

<!-- NAGŁÓWEK -->
<entity name="B1500A" class="ptolemy.moml.EntityLibrary">
  <configure>
    <?moml
      <group>
        <doc>B1500A analyzer actors </doc>

        <!-- INFORAMCJA O KLASIE NOWEGO AKTORA -->
        <entity name="b1500_setVoltage"
        class="ptolemy.actor.lib.b1500_setVoltage"/>
        <entity name="b1500_openChannel"
         class="ptolemy.actor.lib.b1500_openChannel"/>
        <entity name="b1500_closeChannel"
         class="ptolemy.actor.lib.b1500_closeChannel"/>
        <entity name="b1500_currentMeasure"
         class="ptolemy.actor.lib.b1500_currentMeasure"/>
        <entity name="b1500_voltageMeasure"
         class="ptolemy.actor.lib.b1500_voltageMeasure"/>
        <entity name="b1500_sweepVoltage"
         class="ptolemy.actor.lib.b1500_sweepVoltage"/>
        <entity name="b1500_setMeasurment"
         class="ptolemy.actor.lib.b1500_setMeasurment"/>
        <entity name="b1500_setTrigger"
         class="ptolemy.actor.lib.b1500_setTrigger"/>
        <entity name="b1500_exTrigger"
         class="ptolemy.actor.lib.b1500_exTrigger"/>
        <entity name="b1500_resetAll"
         class="ptolemy.actor.lib.b1500_resetAll"/>
        <entity name="b1500_getData"
         class="ptolemy.actor.lib.b1500_getData"/>
        <entity name="b1500_runMeasure"
         class="ptolemy.actor.lib.b1500_runMeasure"/>
      </group>
```

```
38      ?>
39    </configure>
40  </entity>
```

Budowa pliku nie jest skomplikowana i jest podobna do opisywanego w rozdziale o plikach konfiguracyjnych. Istotna jest część z widocznymi tagami o nazwie *entity*. Najpierw poprzez parametr *name* ustawiana jest nazwa wyświetlana aktora, następnie podawana jest nazwa klasy w której znajduje się ciało aktora (klasa z pliku Java). Zaprezentowany plik konfiguracyjny nazywa się *b1500.xml*(od nazwy analizatora B1500A). Informacje o nim zawarte są w pliku *devices.xml*, by ostatecznie informacje umieszczone zostały w pliku *actors.xml*.

Dzięki takiej strukturze informacji w plikach konfiguracyjnych, nowy aktor pojawi się w odpowiedniej zakładce, zgodnej z tym gdzie został umieszczony. Od tej pory może być używany do budowy modeli pomiarowych. Menu z nowym aktorem zaprezentowano na rysunku 4.7.

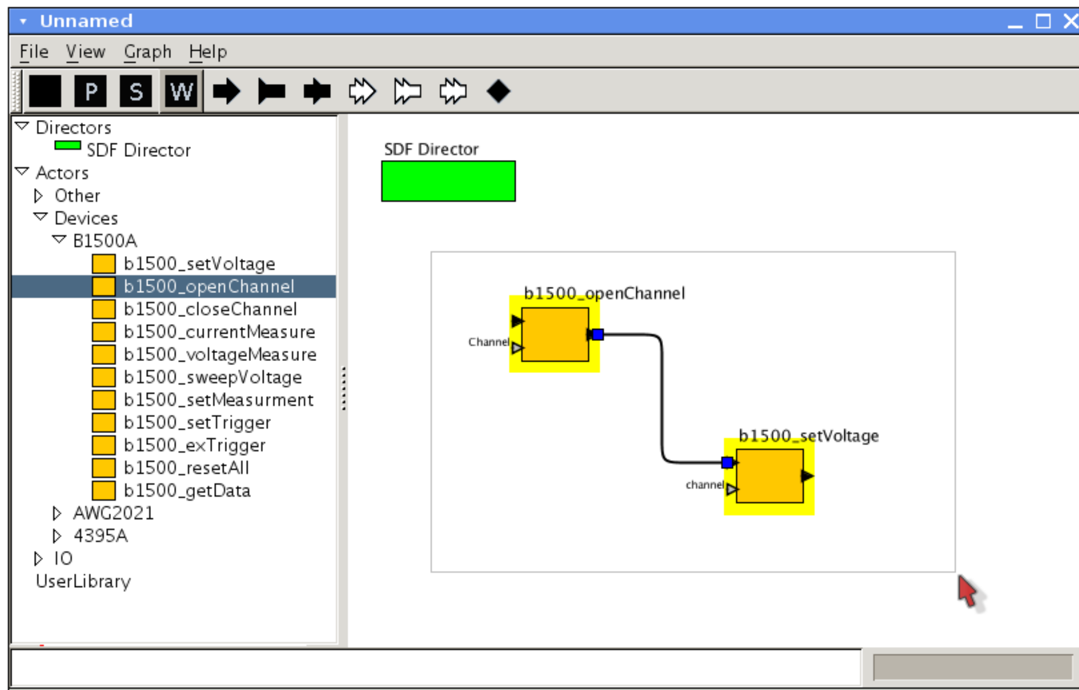


Rysunek 4.7. *Po stworzeniu pliku Java i umieszczeniu informacji w pliku XML, aktor pojawia się w odpowiednim menu.*

Dzięki takiej konstrukcji, łatwo można utrzymać porządek przy dużej liczbie aktorów (do tej pory stworzono ich około 60-ciu), segregując je w zależności od zastosowań, czy przynależności do urządzeń.

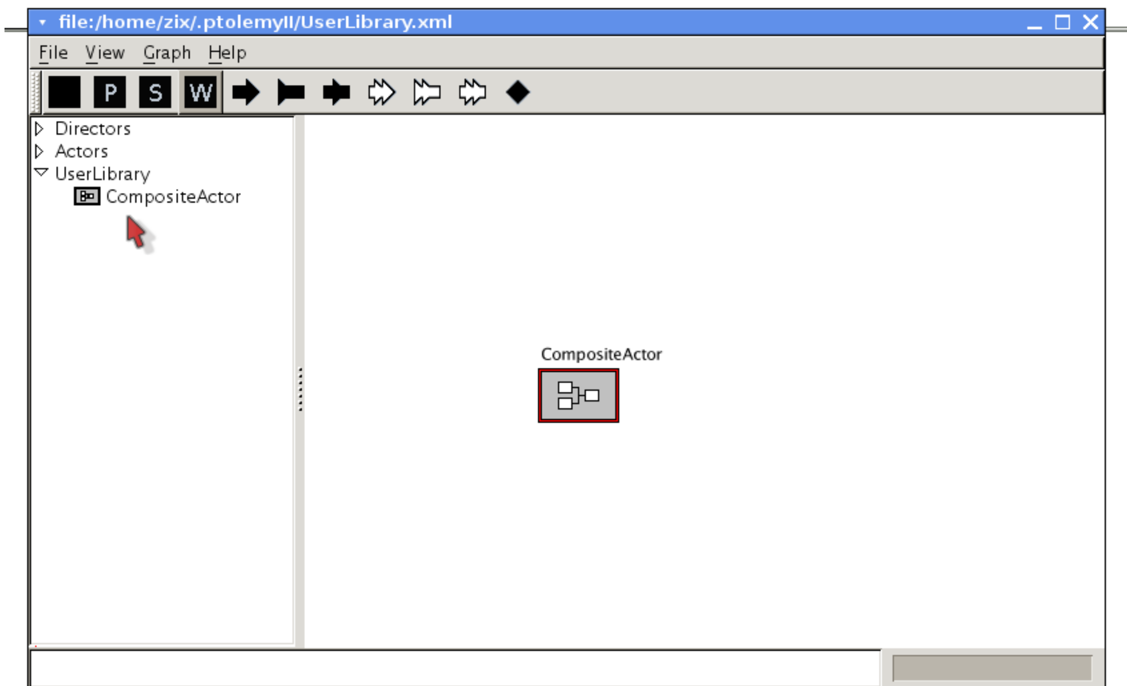
4.4.2. Tworzenie aktora poprzez edytor graficzny z aktorów już istniejących

Przytoczony sposób przedstawia budowę aktora poprzez plik Java, co umożliwia nadanie mu wręcz nieograniczonej funkcjonalności (użytkownika ograniczają tylko aspekty techniczne języka Java i wyobraźnia). Możliwe jest także zbudowanie aktora z aktorów gotowych, dostępnych w środowisku. Wykorzystując ich prostą funkcjonalność, buduje się aktorów do celów bardziej złożonych (coś na wzór małych modeli). W tym celu należy umieścić w modelu aktorów z których zbudowany ma być nowy aktor. Określić wszystkie zależności pomiędzy nimi. Dodatkowo z menu górnego umieścić można porty, które będą portami wejściowymi i wyjściowymi nowego aktora. Następnie zaznaczyć całość i z menu „VIEW" wybrać opcję „COMPOSITE ACTOR" (istnieje także skrót klawiszowy do tego celu „CTRL-A"). Dla lepszego zobrazowania tego procesu przedstawiono taką sytuację na rysunku 4.8.



Rysunek 4.8. *W celu stworzenia aktora, należy zaznaczyć obszar z aktorami, które stanowią jego wnętrze. Następnie wybrać z menu „VIEW" opcję „COMPOSITE ACTOR".*

Gdy zostanie wybrana wspomniana opcja, pojawi się nowe okno modelu z umieszczonym w nim aktorem o nazwie „composite actor". Nazwę zmienić można wchodząc w ustawienia aktora i wybierając „Rename actor". Należy jeszcze zapisać aktora wciskając CTRL-S i wybierając nazwę pliku. Od tej pory nowy aktor dostępny jest poprzez zakładkę „USER LIBRARY" w oknie po lewej stronie. Zakładkę tę zaprezentowano na rysunku 4.9.



Rysunek 4.9. *Każdy stworzony aktor dostępny jest poprzez „USER LIBRARY".*

Wnętrze aktora można edytować wybierając w ustawieniach opcję „open actor”. Otwiera się wtedy nowe okno ze strukturą wewnętrzną elementów składających się na aktora. Dzięki tej opcji tworzenie modeli staje się dużo szybsze i efektywniejsze, gdyż użytkownik w prosty sposób może budować aktorów o funkcjonalności prostych modeli i korzystać z nich w przyszłości. Z czasem dzięki temu narzędziu, użytkownik sam może rozszerzać bazę aktorów, nie będąc zmuszonym do oprogramowywania nowych aktorów.

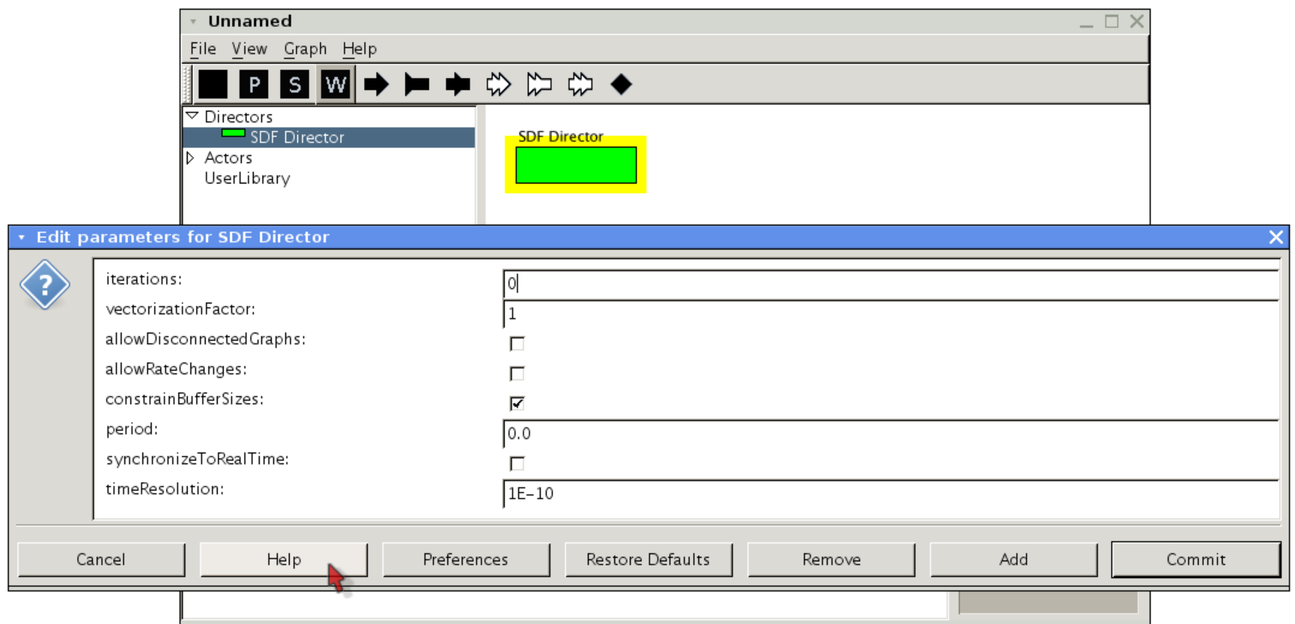
Podsumowując, istnieją dwa sposoby tworzenia aktorów. Pierwszy nadaje się do dodawania aktorów o funkcjonalności podstawowej. Na przykład aktor do obsługi nowego urządzenia etc. Natomiast drugi sposób jest wygodniejszy do pracy na już gotowych aktorach, na przykład w celu tworzenia bardziej wyspecjalizowanych aktorów reprezentujących różne urządzenia pomiarowe i ich funkcje.

4.4.3. Reżyser

O Reżyserach wspomniano króciutko w jednym z rozdziałów powyżej. Zostały one odziedziczone po środowisku Ptolemeusz i każdy budowany model musi w swojej strukturze posiadać Reżysera. Są to na dobrą sprawę aktorzy o ogólniejszych uprawnieniach względem modelu. W oryginalnej wersji oprogramowania, czyli Ptolemeuszu, ich znaczenie jest bardzo duże, ponieważ narzędzie to służy do symulacji różnych procesów. Ważna jest więc na przykład forma przedstawiania czasu, czy specyficzny sposób interakcji pomiędzy aktorami. Natomiast projektowane środowisko GUI opiera się na pracy z urządzeniami fizycznymi, co mocno ogranicza zastosowanie Reżyserów. Przepływ czasu z góry jest określony (czas rzeczywisty), nie ma także dużego pola manewru jeśli chodzi o rodzaje modeli, gdyż każdy jest w pewien sposób wirtualną reprezentacją rzeczywistych połączeń pomiędzy urządzeniami. Z tego też powodu w środowisku istnieje tylko jeden rodzaj Reżysera, który operacje wykonuje sekwencyjne. Pozwala on na ustawienie parametrów takich jak liczba powtórzeń modelu (iteracji), czy czas który ma upłynąć pomiędzy uruchomieniem kolejnych aktorów. Posiada także opcję, która umożliwia umieszczenie w modelu aktorów nie połączonych z innymi aktorami, gdyż domyślnie umieszczenie aktora nie połączonego z innymi poprzez jakikolwiek port jest zabronione. Na rysunku 4.10 zaprezentowano opcje, które użytkownik może zmieniać.

Bardzo ważną opcją jest *iterations*, gdyż umożliwia tworzenie modeli z parametrami, które ewoluują w czasie. Jest to niezmiernie przydatne przy pracy z urządzeniami takimi jak zasilacze, gdzie często pożądane jest przemiatanie napięcia w szerszym zakresie. Dzięki owej opcji proces ten może zostać zautomatyzowany (przykład modelu korzystającego z takiej iteracji dostępny jest w rozdziale opisującym pomiary I-V i C-V sensorów).Opcja ta powoduje wykonanie modelu zadana liczbe razy. Kolejnymi parametrami są *allowDosconectedGraph*, gdy opcja ta jest aktywna, możliwe jest umieszczanie aktorów bez połączenia z innymi. Wreszcie możliwe jest ustawienie czasu jaki ma upłynąc pomiezy uruchomieniem kolejnych aktorów. Jest wbrew pozorom niezmiernie ważne, gdyż niektóre urządzenia potrzebują okresloną ilość czasu aby ustawić wybrane opcje. Z problemem tym zetknięto się na przykład podczas prac z analizatorem widma HP4395A, który potrzebuje około jedno sekundowej przerwy po pomiarze aby poprawnie załadować dane pomiarowe do swojej wewnętrznej pamięci.

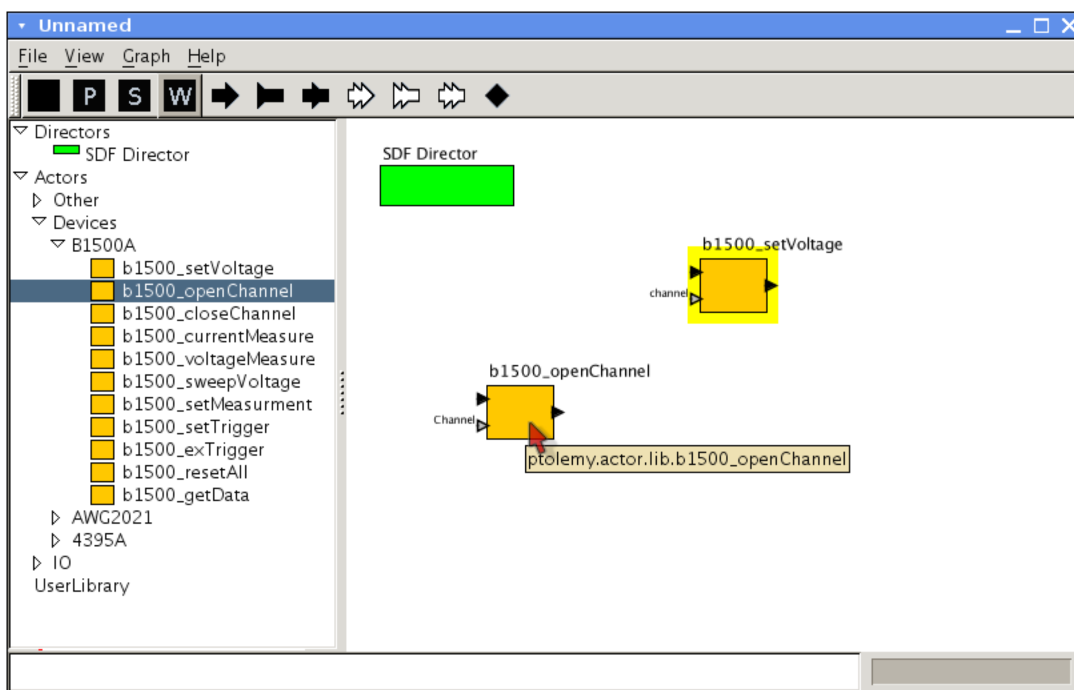
Podsumowując, w środowisku zostawiono jedynie jednego reżysera, a dodatkowo ograniczono jego funkcjonalność. Uczyniono tak aby dostosować ten element do stawianych założeń. Rozważana była także możliwość stworzenia reżysera, który zarządzał by aktorami w sposób równoległy. Poczyniono w tym kierunku pewne prace, ale projekt ten ostatecznie zostawiony został jako element dalszego rozwijania środowiska.



Rysunek 4.10. *Opcje dla reżysera.*

4.5. Tworzenie modelu

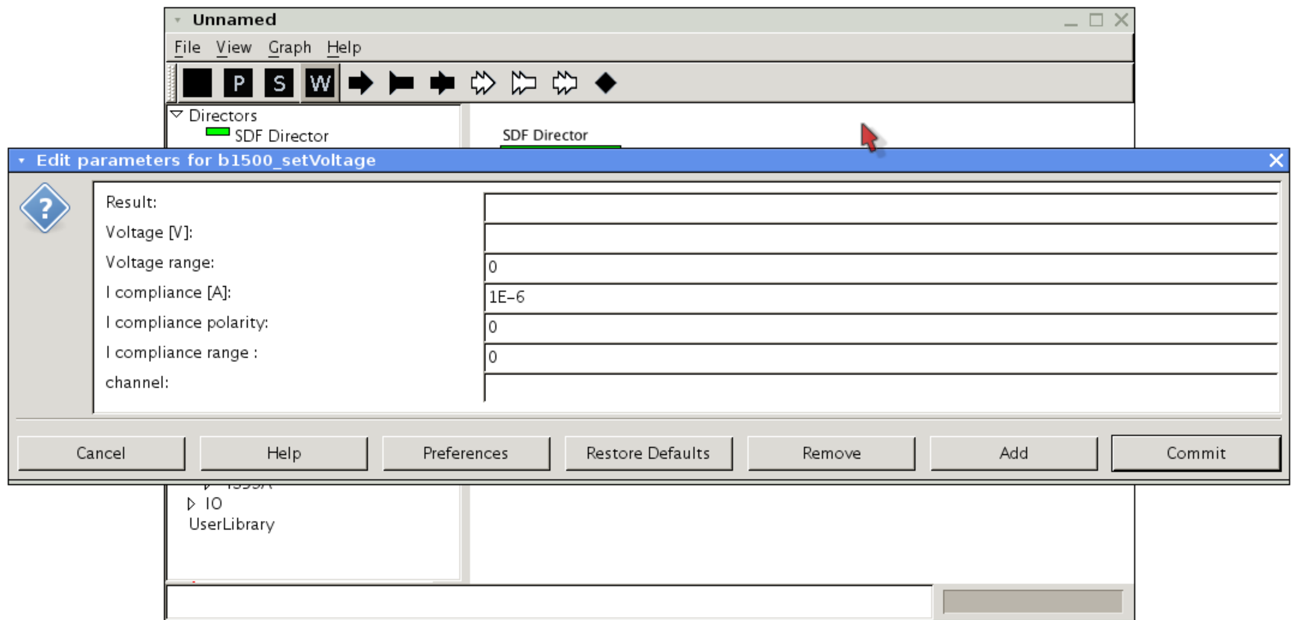
Po przybliżeniu pojęcia aktora i reżysera, a także po opisaniu podstawowej funkcjonalności środowiska opisać można proces tworzenia modelu. Otóż pierwszą czynnością powinno być umieszczenie w modelu reżysera, a także wybranych aktorów, co zaprezentowano na rysunku 4.11.



Rysunek 4.11. *Ogólna struktura środowiska GUI w oparciu o narzędzie Ptolemeusz.*

W reżyserze należy ustawić jego podstawowe parametry, to znaczy ilość iteracji i czas jaki ma

upłynąć pomiędzy wykonaniem kolejnych aktorów (ma to znaczenie przy niektórych urządzeniach, które nie mogą dostawać komend zbyt szybko). Po określeniu tych parametrów można zacząć budować połączenia pomiędzy aktorami. Dobór aktorów jest oczywiście uzależniony od tego, co model ma reprezentować. Po umieszczeniu wszystkich elementów należy ustawić odpowiednie parametry. Dostęp do okna opcji uzyskuje się poprzez dwukrotne kliknięcie w wybrany element (aktora/reżysera). Widok na przykładowe okno parametrów przedstawiono na rysunku 4.12.

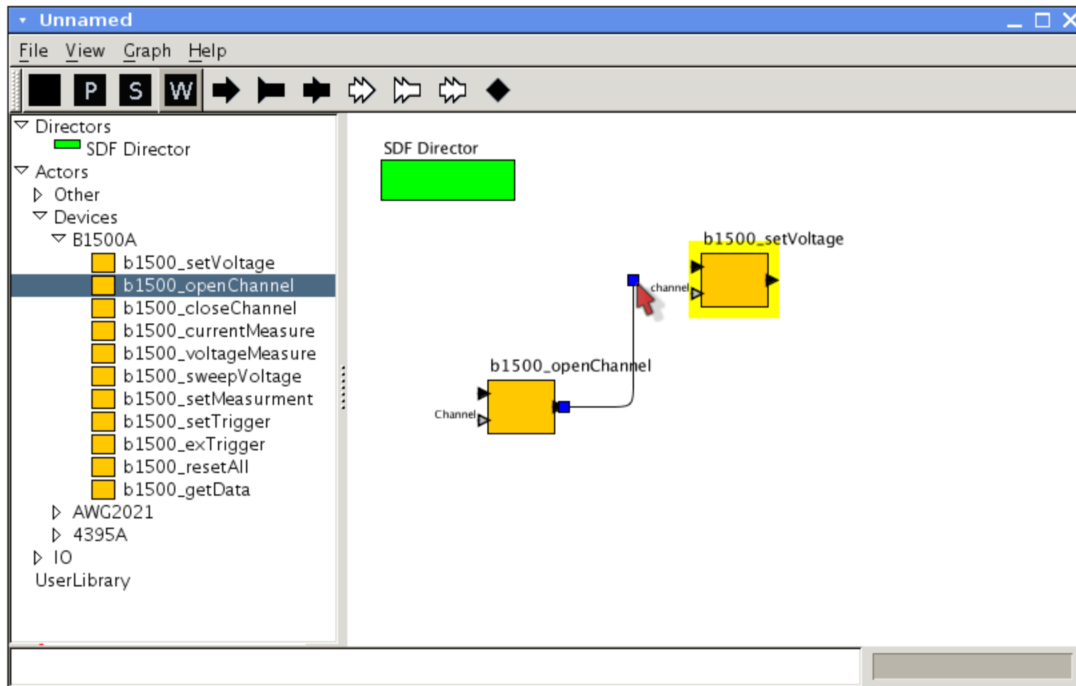


Rysunek 4.12. *Przykładowy widok na okno opcji dla wybranego aktora.*

Na przytoczonym przykładzie istnieje sześć parametrów. Cztery z nich posiadają domyślne wartości, oczywiście użytkownik może je zmieniać, ale można także pozostać przy tych wartościach. Kolejne dwa parametry nie posiadają przypisanej wartości, co oznacza, iż użytkownik jest zobligowany do umieszczenia tam odpowiednich wartości, jeżeli tego nie zrobi środowisko zgłosi błąd. Po określeniu parametrów należy zaakcepotwać zmiany kliknięciem w przycisk „commit”.

Po ustawieniu parametrów konieczne jest odpowiednie połączenie portów wyjściowych i wejściowych, tak aby otrzymać żądany efekt kolejności uruchamiania aktorów. Aby połączyć porty należy złapać, wciskając lewy przycisk myszy, port z którego sygnał ma wychodzić, a następnie, nie puszczając przycisku, przeciągnąć pojawiającą się „linkę” do portu na który sygnał ma być wysyłany. Najczęściej górny port wejściowy i wyjściowy służą do hierarchizacji w modelu i wyznaczenia kolejności uruchamiania aktorów. Należy jednakże zwrócić uwagę na to co i czy dany port przekazuje jakieś informacje. Można to sprawdzić najeżdżając myszką na port, po chwili pojawia się chmurka z opisem portu, gdzie zawarte są informacje o typie przekazywanych informacji. Poprzez porty nad którymi istnieje podpis(na przykład „channel” etc.) przekazywać można wartości parametrów. Dzięki temu przekazana wartość wymusza zachowanie aktora, co jeszcze bardziej rozszerza możliwości modelowania.

Przykładowo dla dwóch aktorów, przedstawiono na rysunku 4.13.



Rysunek 4.13. *Łączenie portów wybranych aktorów.*

Po ustaleniu połączeń model gotowy jest do uruchomienia. Aby uruchomić model należy wcisnąć P na pasku górnym. Na liście komend na dole okna pojawi się napis „executing". Model skończył się wykonywać jeżeli pojawi się napis „finished". Forma przedstawienia wyników zależna jest od tego jaki aktor ma je reprezentować. Składa się na to także cała struktura przekazywanych danych. W środowisku stworzono szereg aktorów których celem jest jedynie obróbka uzyskanych informacji. Są to na przykład proste operacje matematyczne, czy operacje na danych tekstowych. Istnieje także szereg aktorów do automatycznej wizualizacji pomiaru, pozwala to na podgląd w czasie rzeczywistym, czy wszystko przebiega pomyślnie. Z jednego z takich aktorów korzystano podczas pomiarów sensorów, gdyż konieczny był tam podgląd w czasie rzeczywistym na wartości badanych parametrów.

Po stworzenie modelu można go oczywiście zapisać celem późniejszego wykorzystania. Jak było to wspomniane wcześniej w pracy, można to zrobić za pomocą skrótu klawiszowego CTRL-S. Model zapisywany jest do pliku XML, z nazwą nadaną przez użytkownika. Późniejsze odczytanie możliwe jest za pomocą skrótu klawiszowego CTRL-O.

Rozdział 5

Przykładowe modele pomiarowe

W rozdziale tym zaprezentowane są przykładowe modele pomiarowe, które wykorzystane zostały do testowych pomiarów wybranych układów elektronicznych i detektorów. Budowane stanowiska pomiarowe wymagały do testów różnych zestawów urządzeń pomiarowych, począwszy od zasilaczy po dokładne mierniki LCR. Celem tej części pracy było sprawdzenie w praktyce napisanego oprogramowania. Wykonywane pomiary były czysto testowe, ale kilka modeli okazało sie na tyle sprawnych, iż za ich pomocą zebrane zostały wyniki, które mogły być użyte do obliczeń wykorzystywanych w publikacjach naukowych. Jak pokazały te próbne pomiary oprogramowanie działa poprawnie, choć podczas testów wyłoniło się kilka obszarów oprogramowanie (dotyczących obsługi środowiska graficznego), które wymagają udoskonalenia.

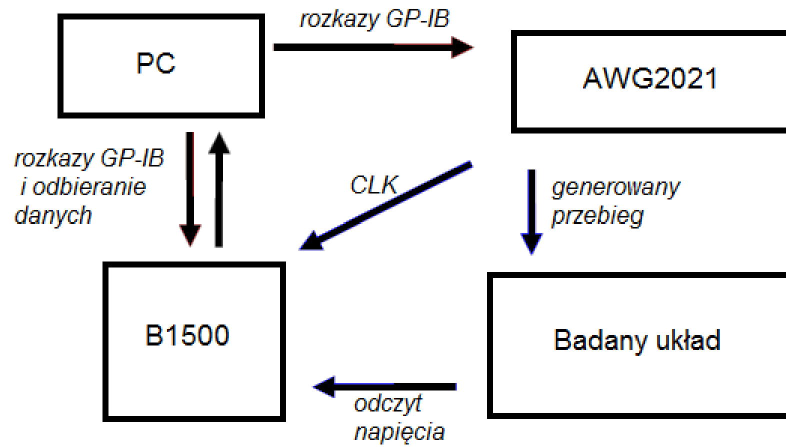
W miejscu tym należy wspomnieć, iż oprogramowanie tworzone było w języku angielskim, aby umożliwić korzystanie z niego cudzioziemcom pracującym w labolatorium. Dlatego też cała komunikacja z użytkownikiem odbywa się w tym języku.

5.1. Stanowisko pomiarowe do pomiarów statycznych i pomiarów mocy układów DAC

W tym przypadku należało zbudować stanowisko pomiarowe, którego celem było wykonanie pomiarów statycznych układu DAC[30] (układ posiada 5 kanałów), zaprojektowanego w zespole Elektroniki Jądrowej na potrzeby przyszłej elektroniki odczytu detektora LumiCal [31] dla eksperymentu ILC [32] (International Linear Collider). Stanowisko składa się z dwóch urządzeń, to jest generatora przebiegów AWG2021 i analizatora urządzeń półprzewodnikowych B1500A. Zadaniem generatora jest dostarczanie przebiegu w postaci RAMPy (odpowiednio zmodyfikowanego, o czym dalej w tym rozdziale), natomiast analizator służy odczytywaniu napięć i prądów. Dodatkowo należało stworzyć całe oprogramowanie wykorzystywane w pomiarach, czyli program sterujący do obsługi urządzenia B1500A, a także odpowiednie oprogramowanie dla AWG2021. Całość zintegrowana została ze środowiskiem graficznym, poprzez które wykonywane były pomiary.

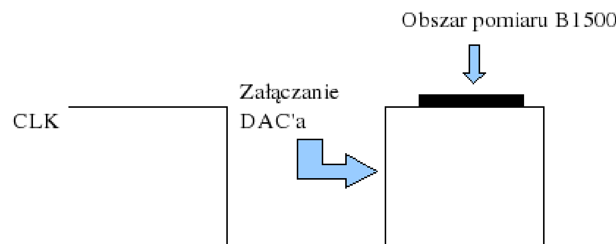
5.1.1. Budowa i oprogramowanie stanowiska

Jak było już wspomniane, stanowisko składa się z dwóch urządzeń pomiarowych, sterowanych za pomocą komputera PC. Schemat stanowiska zaprezentowany jest na rysunku 5.1.



Rysunek 5.1. *Schemat stanowiska do pomiarów statycznych i pomiarów mocy układów DAC.*

Na generator AWG2021 ładowany jest przebieg w postaci odpowiednio zmodyfikowanej RAMPy (modyfikacja wymagana jest przez układ DAC, który posiada rozdzielony stan 511). Generator posiada wyjście cyfrowe, podające kolejne stany cyfrowe, przy kolejnych cyklach zegara. Wyjście zegara generatora AWG2021 połączone zostało z wejściem triggera (sygnału wyzwalającego) zewnętrznego analizatora B1500A. Następnie poprzez oprogramowanie analizator konfigurowany jest w taki sposób, aby pomiar wykonywany był przy wysokim stanie zegara. Zobrazowano to na rysunku 5.2.



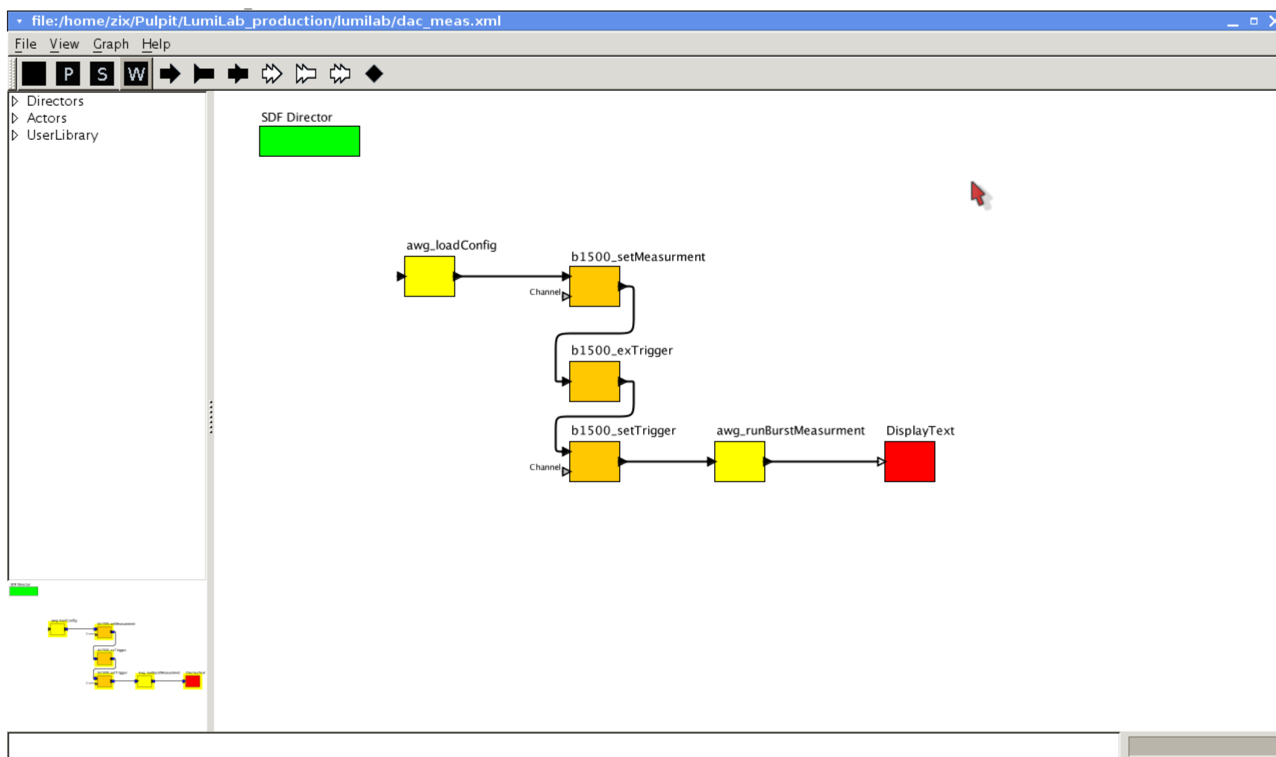
Rysunek 5.2. *Cykle zegara generowanego poprzez AWG z zaznaczonym miejscem pomiaru analizatora B1500A.*

Wspomnieć należy, iż szybkość pomiaru analizatora ogranicza częstotliwość taktowania zegara do 50Hz. Powyżej tej wartości miarodajny pomiar jest niemożliwy (analizator nie zapisuje do pamięci kolejnych pomiarów). Ograniczenie to jednak nie wpływa znacząco na pomiary statyczne, gdzie 50Hz jest wartością odpowiednią.

Standardowo, poprzez stworzone oprogramowanie sterujące mierzyć można napięcie na wyjściu DACa, wtedy otrzymuje się charakterystykę statyczną. Możliwa jednakże jest też taka konfiguracja, gdzie korzystając z wszystkich czterech kanałów (SMU) analizatora, określić można moc na wybranym kanale DAC (układ DAC posiada pięć kanałów). W konfiguracji takiej prowadzony jest pomiar prądu i na tej podstawie obliczana jest szukana moc. W pomiarach tych wykorzystywany jest jedynie analizator B1500A, pomiar prowadzony jest na wszystkich czterech kanałach, celem uzyskania danych do wyznaczenia mocy.

5.1.2. Model wykorzystywany w pomiarach

Model składa się z sześciu aktorów, dwóch dla generatora AWG2021, trzech dla analizatora B1500A i jednego aktora pomocniczego do wyświtlania tekstu. Model ten zaprezentowano na rysunku 5.3.

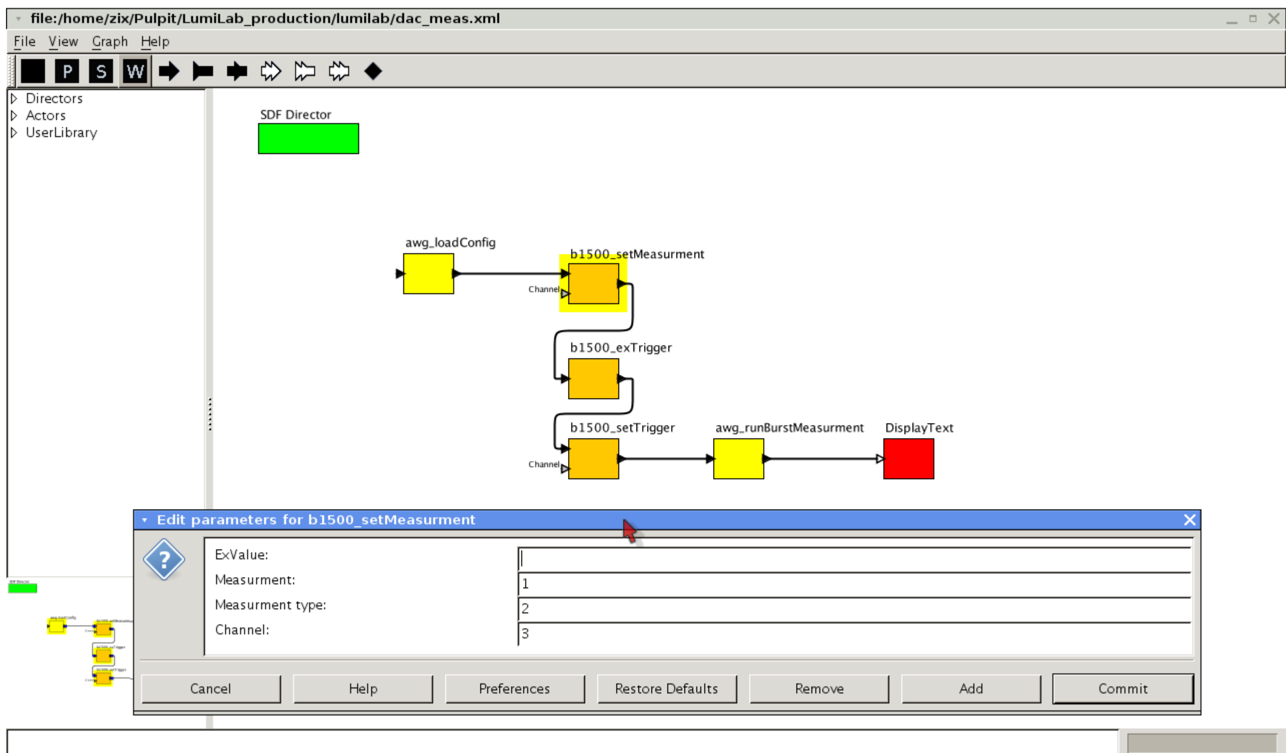


Rysunek 5.3. *Model do pomiarów statycznych układów DAC.*

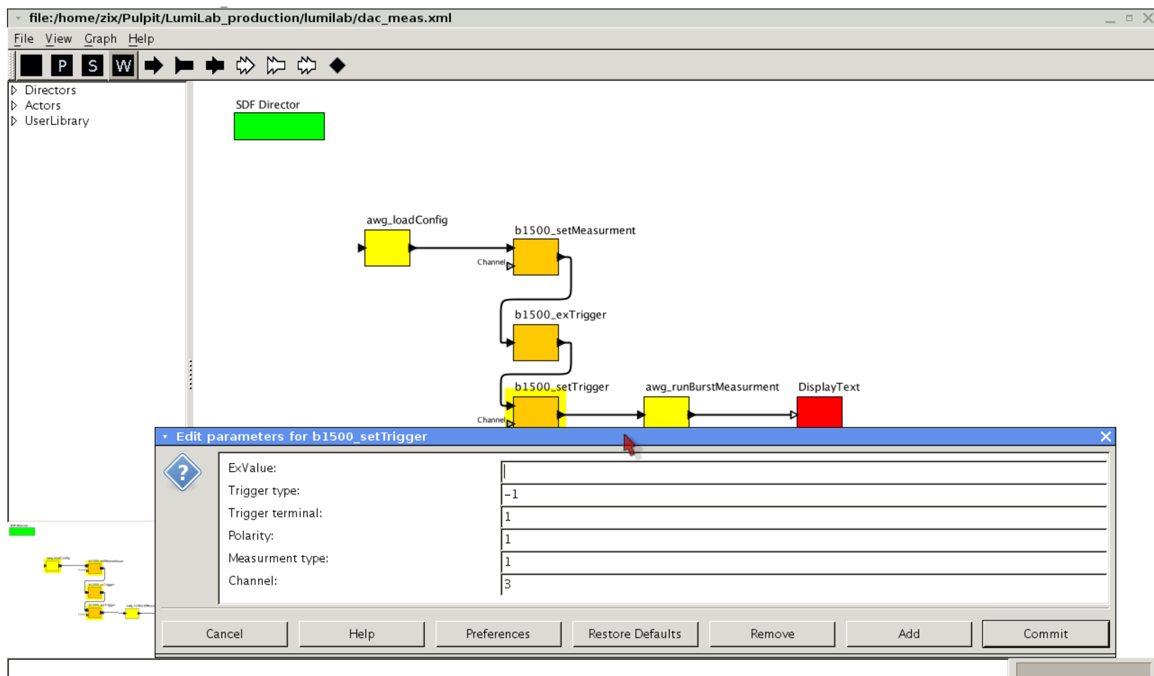
Z zaprezentowanych aktorów, dwóch wymaga nastawienia parametrów przez użytkownika. Pierwszy to aktor o nazwie „setMeasurement”, w którym ustawiać można parametry, które widoczne są na rysunku 5.4. Wartości owych parametrów zaczerpnięte są z podręcznika dla analizatora. Pozwalają one na skonfigurowanie typu pomiaru (parametr measurement), to znaczy czy ma być to pomiar pojedyńczy, wielokrotny, wymuszony, a także rodzaju pomiaru (parametr *measurement type*): prądowy-napięciowy i na końcu, na którym SMU pomiar będzie prowadzony (parametr *channel*).

Drugim aktorem o nastawianych parametrach jest aktor „setTrigger”, a jego parametry przedstawiono na rysunku 5.5. Każdy z tych parametrów posiada wartości domyślne. W tym przypadku parametry znaczą kolejno: typ trigera (zewnętrzny/wewnętrzny), terminal triggera (wyjście/wejście), jaki sygnał ma wyzwalać trigger (dodatni/ujemny), rodzaj pomiaru (napięciowy/prądowy) i podobnie jak wyżej numer SMU. Z racji tego, iż wszystkie parametry są domyślne, jeżeli nie ma potrzeby, użytkownik nie powinien ich zmieniać. Gdyby jednak zaistniała konieczność ich zmiany, opis znajduje się w podręczniku programistycznym dla urządzenia przy opisie komendy, TGPS.

Dodatkowo aktor dla generatora AWG2021 o nazwie „loadConfig” posiada parametr w którym podaje się nazwę pliku z przebiegiem, który ma być generowany (jak wykonać taki plik opisane jest w podręczniku dla tego generatora). Pozostałych dwóch aktorów nie posiada parametrów do nastawiania. Pierwszy z nich o nazwie „exTrigger” ustawia analizator w tryb zewnetrznego wyzwalania. Drugi natomiast, o nazwie „runBurstMeasurement” uruchamia po-



Rysunek 5.4. *Parametry ustawiane przez użytkownika dla aktora „setMeasurement”.*

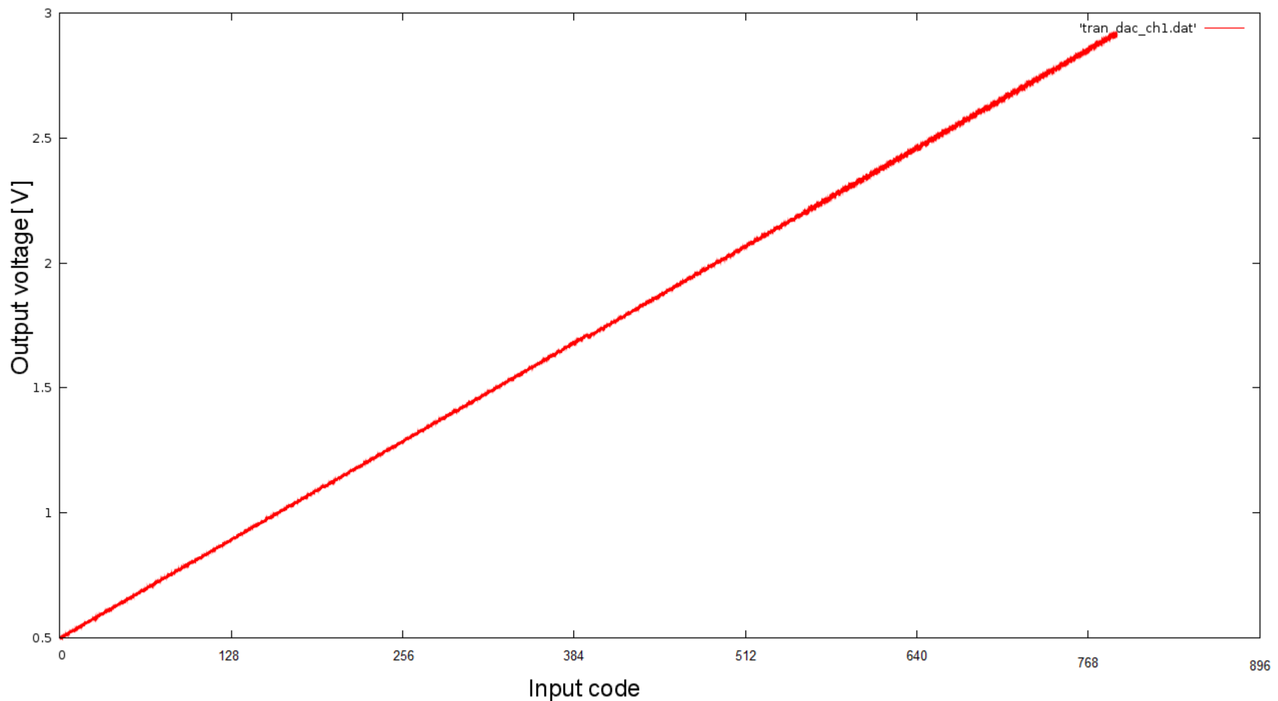


Rysunek 5.5. *Parametry ustawiane przez użytkownika dla aktora „setTrigger”.*

miar. Po zakończeniu działania informacje zapisywane są do plików o nazwie „dataXX.out” gdzie XX to kolejne numery 01,02 etc. Dane te reprezentują wykonane pomiary (napięcia/prądy). Mogą być następnie dalej przetwarzane, na przykład w Gnuplocie, czy innych programach analizy danych.

5.1.3. Wyniki

Wyniki zebrane zostały dla wszystkich pięciu kanałów, w celu określenia ewentualnych rozrzutów pomiędzy nimi. Zebrano charakterystyki statyczne, oraz określono moc na jednym z kanałów (dla przykładu). Na rysunku 5.6 przedstawiono przykładowe wyniki uzyskane dla badanego układu, dla kanału numer jeden.



Rysunek 5.6. *Funkcja przenoszenia dla kanału numer 1 badanego układu.*

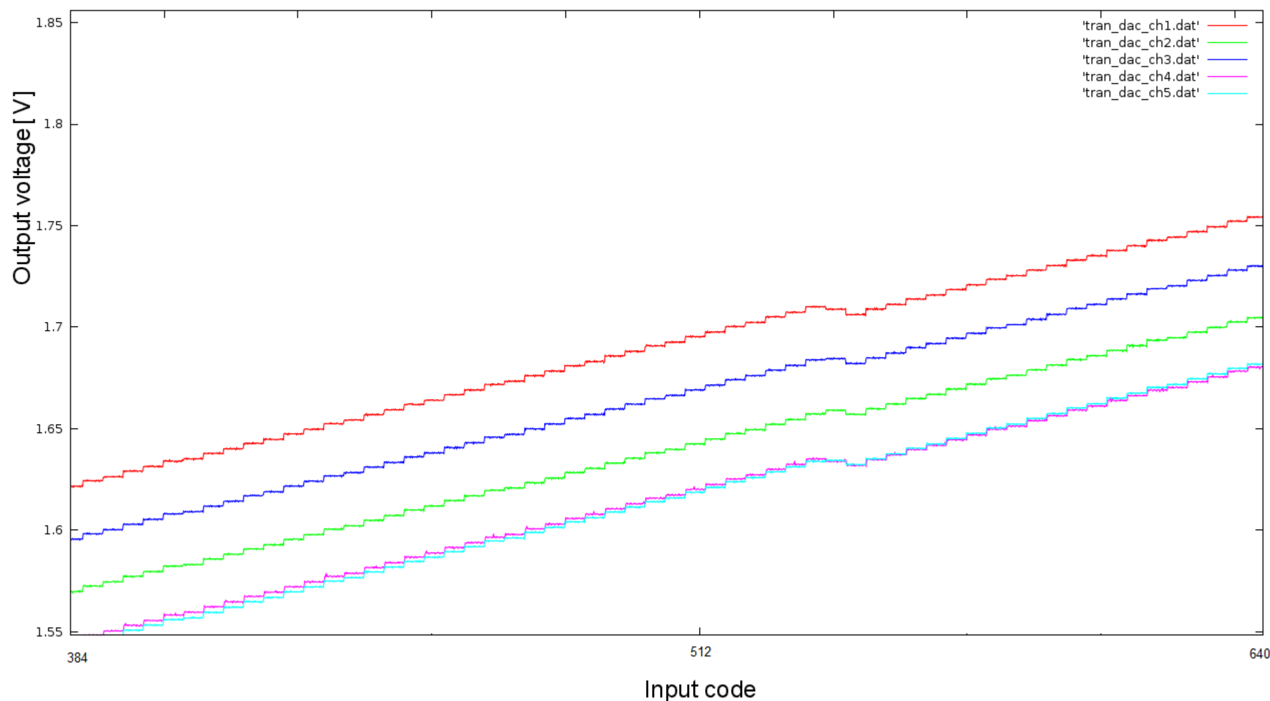
Natomiast na rysunku 5.7 zaprezentowane są wyniki dla wszystkich pięciu kanałów. Na rysunku tym lepiej widać zmieniające się stany, gdyż zawężono pokazywany obszar. Na rysunku pierwszym(5.6) pokazana jest całość pomiaru, co powoduje iż „schodki" się zlewają. Ponieważ wyniki zbierane są w formie pliku tekstowego, możliwa jest ich dalsza obróbka w celu uzyskania dodatkowych informacji.

Dzięki elastycznej budowie środowiska funkcje do obróbki danych można łatwo podpiąć do środowiska graficznego, co w przyszłości może zaowocować prostszą analizą danych. Poza pomiarami funkcji przenoszenia, wykonano także przykładowe pomiary mocy dla jednego kanału. W tabeli 5.1 przedstawiono zebrane wyniki.

Stan	0	128	256	511	1022
Moc [μW]	450	445	490	615	608

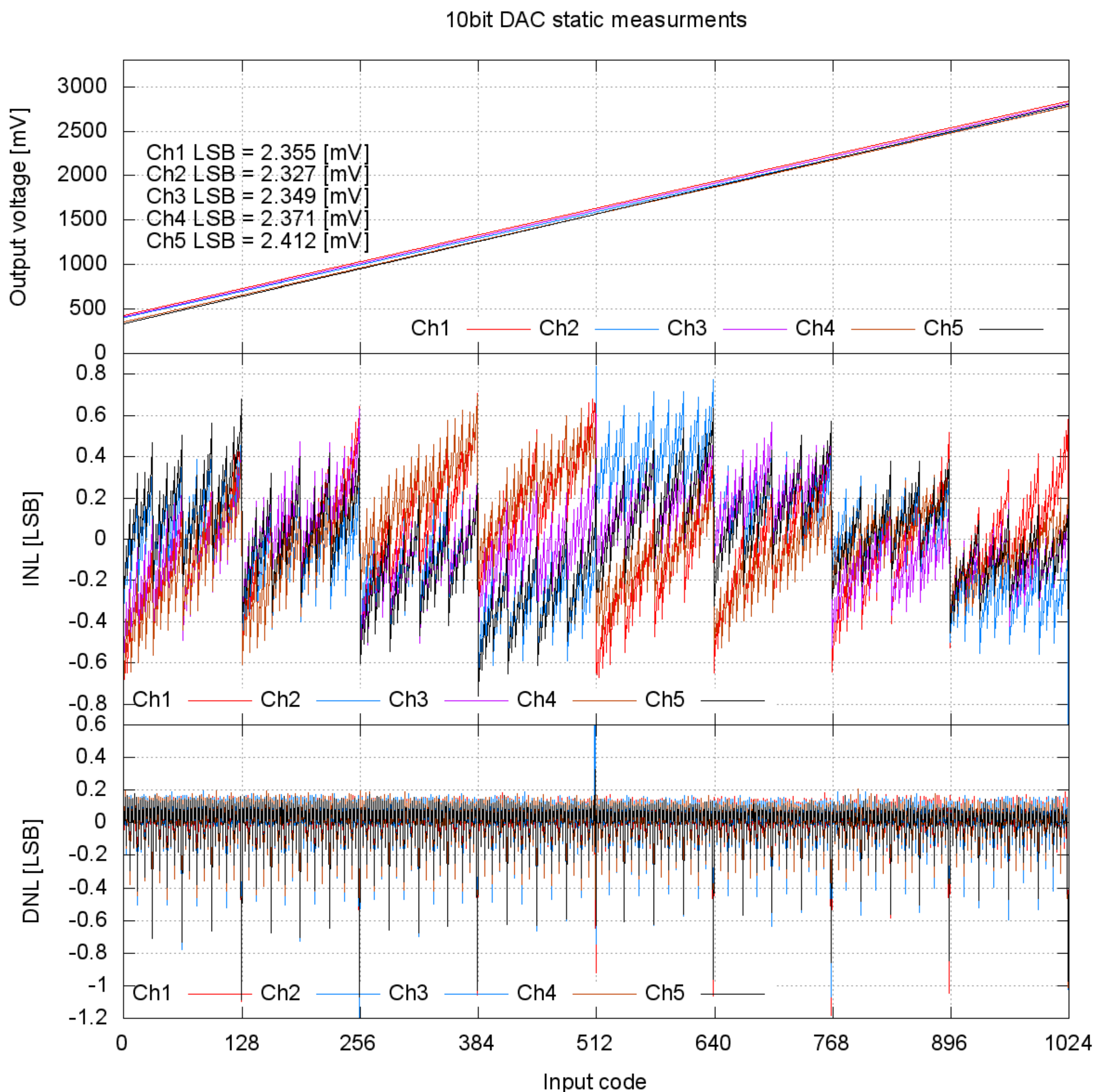
Tablica 5.1. *Przykładowe wyniki pomiarów mocy za pomocą stworzonego oprogramowania.*

Dodatkowo na rysunku 5.8 przedstawiono przykładową obróbkę danych. Celem było określenie nieliniowości różniczkowej (DNL) i całkowej (INL).



Rysunek 5.7. *Wyniki zbiorcze dla wszystkich pięciu kanałów. Wykres przedstawia jedynie wycinek z całości zebranych danych.*

Podsumowując, stworzone stanowisko pomiarowe pozwoliło na dość sprawne wykonanie założonych pomiarów. Dzięki skorzystaniu z budowanego środowiska graficznego istnieje możliwość dalszej rozbudowy o nowe funkcje. Zaprezentowane wyniki są tylko przykładami, ale wykonane zostały także kompletne pomiary celowe, dla potrzeb analizy parametrów badanego układu. Dzięki zastosowaniu wybranych technologi całość pomiaru została znacznie przyspieszona.



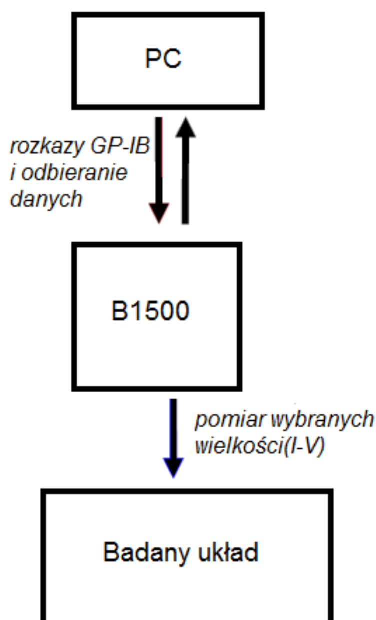
Rysunek 5.8. *DNL i INL dla wybranych kanałów.*

5.2. Pomiary I-V i C-V krzemowych sensorów firmy Hamamatsu

Celem pomiarów było określenie parametrów sensorów wykonanych przez japońską firmę Hamamtsu[33]. Określone miały zostać charakterystyki I-V i C-V, czyli zależność prądu upływu i pojemności w funkcji napięcia polaryzującego. Celem było porównania ich z danymi dostarczonymi przez producenta. Sensory owe wykorzystywane są wspolnie przez Katedrę Oddziaływan i Detekcji Cząstek, oraz Instytut Fizyki Jądrowej w Krakowie do budowy detektorów świetlności dla ILC. Układ zbondowany był w laboratorium mikroelektroniki zespołu Elektroniki Jądrowej WFiIS.

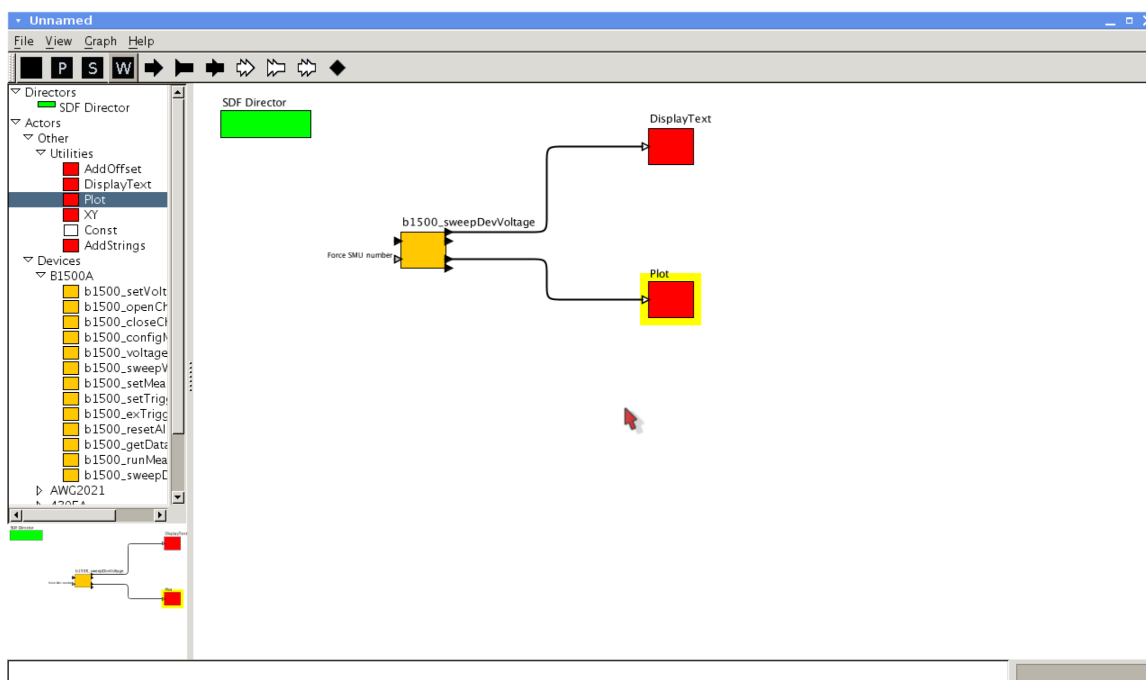
5.2.1. Model do pomiarów I-V krzemowych sensorów firmy Hammamatsu

Pomiary przeprowadzane były za pomocą urządzenia B1500A. Ogólny schemat stanowiska pomiarowego zaprezentowano na rysunku 5.9.



Rysunek 5.9. *Schemat stanowiska pomiarowego do pomiarów I-V sensorów.*

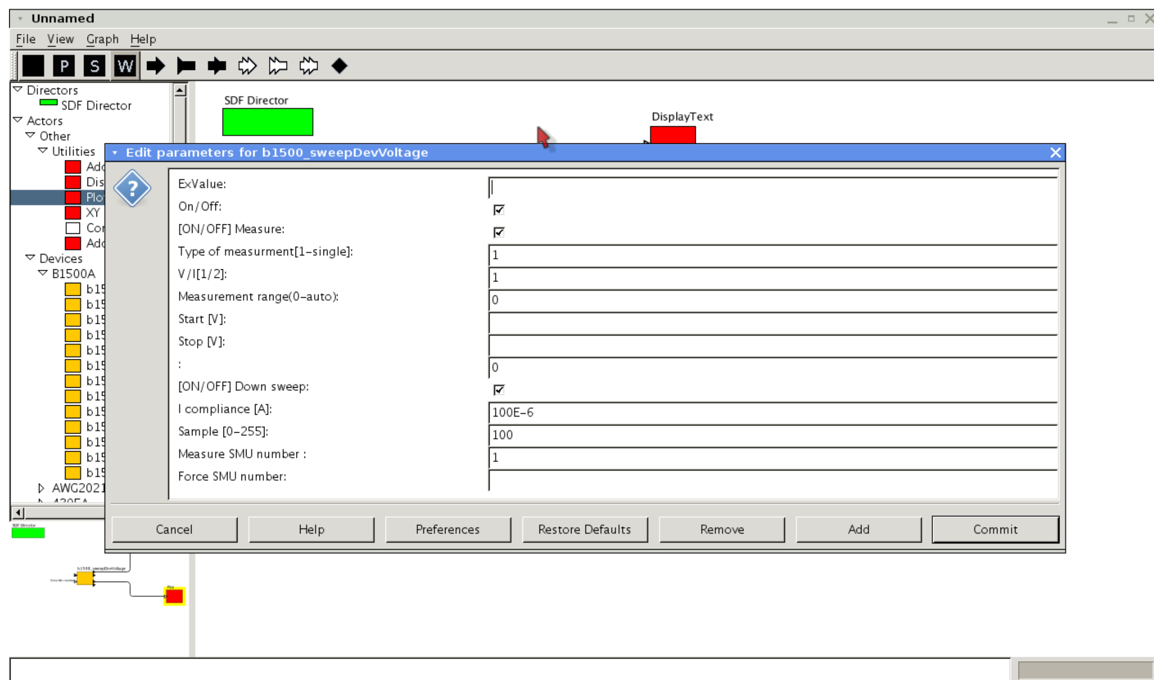
W celu usprawnienia pomiarów stworzony został model pomiarowy w środowisku graficznym, który zaprezentowany jest na rysunku 5.10.



Rysunek 5.10. *Model wykorzystywany w pomiarach I-V sensorów.*

Istniały dwie koncepcje pomiaru, pierwsza zakładała wykorzystanie trybu pojedyńczych pomiarów i manualnego przemiatania napięcia za pomocą oprogramowania. Niestety takie rozwiązanie napotkało na szereg problemów (urządzenie podczas zmiany trybu pracy z napięciowego na prądowy samo zmieniało napięcia co fałszowało wyniki), dlatego też napisano oprogramowanie wykorzystujące tryb pracy urządzenia z przemiatanymi napięciami. Następnie dane odczytywane były na komputerze PC.

Aktor „sweepDevMeasure” wykonujący pomiary posiada kilka parametrów, które nastawić może użytkownik, przedstawiono je na rysunku 5.11.

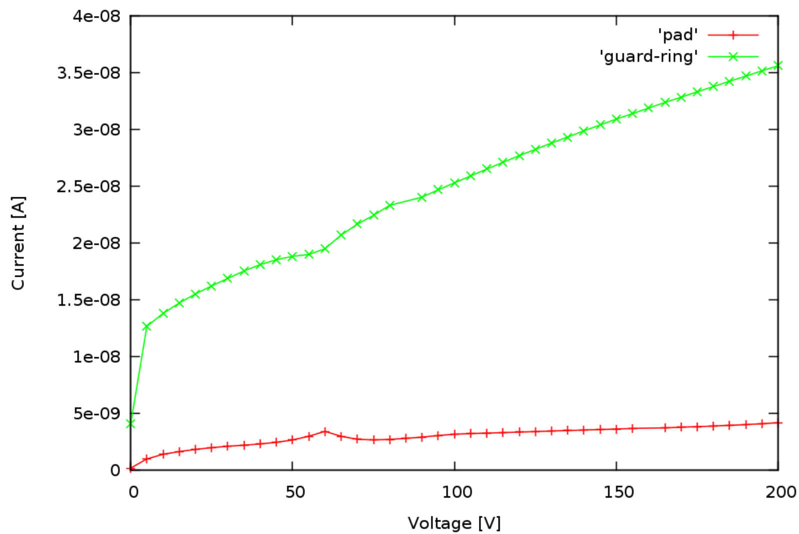


Rysunek 5.11. *Parametry do nastawiania przez użytkownika dla aktora „sweepDevMeasure”.*

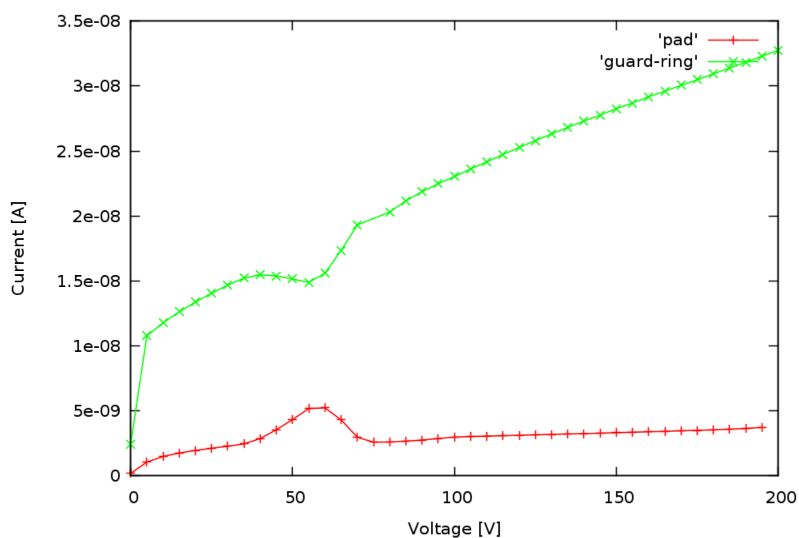
Dodatkowo podczas przeprowadzania pomiarów wyniki w czasie rzeczywistym wykreślane są na wykresie (w tym celu napisano odpowiedniego aktora). Funkcjonalność taką zaimplementowano aby użytkownik mógł kontrolować, czy pomiar przebiega w prawidłowy sposób. Następnie, po wykonaniu pomiaru wyniki zapisywane są do pliku tekstowego celem dalszej obróbki.

5.2.2. Wyniki

Pomiary prowadzone były dla dwóch padów detektora o numerach 63 i 64. Uzyskane przykładowe wyniki zaprezntowano na rysunkach 5.12 i 5.13.



Rysunek 5.12. *Wyniki dla padu numer 63.*



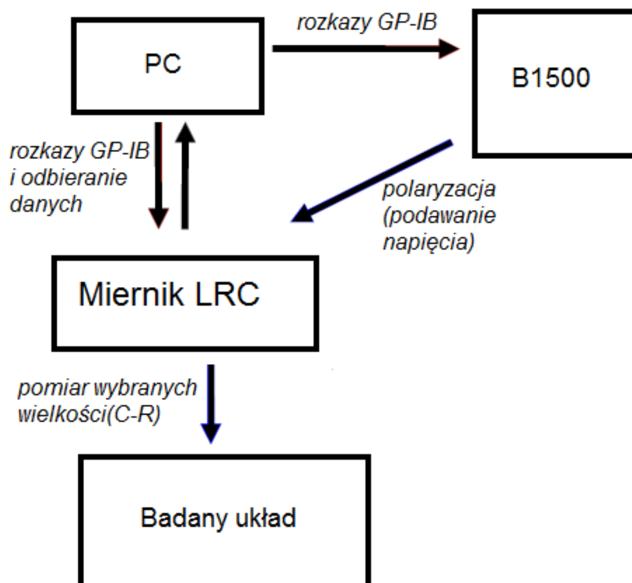
Rysunek 5.13. *Wyniki dla padu numer 64.*

Wyniki zobrazowano za pomocą zewnętrznego programu Gnuplot. Wartości uzyskane są zgodne z tym co w swoich materiałach podaje producent. Dodatkowo zaobserwowano upływ prądu z guard-ringu kosztem padu, czego także się spodziewano. Wyniki te zostały zaprezentowane na spotkaniu kolaboracji FCAL.

5.2.3. Model do pomiarów C-V krzemowych sensorów firmy Hammamatsu

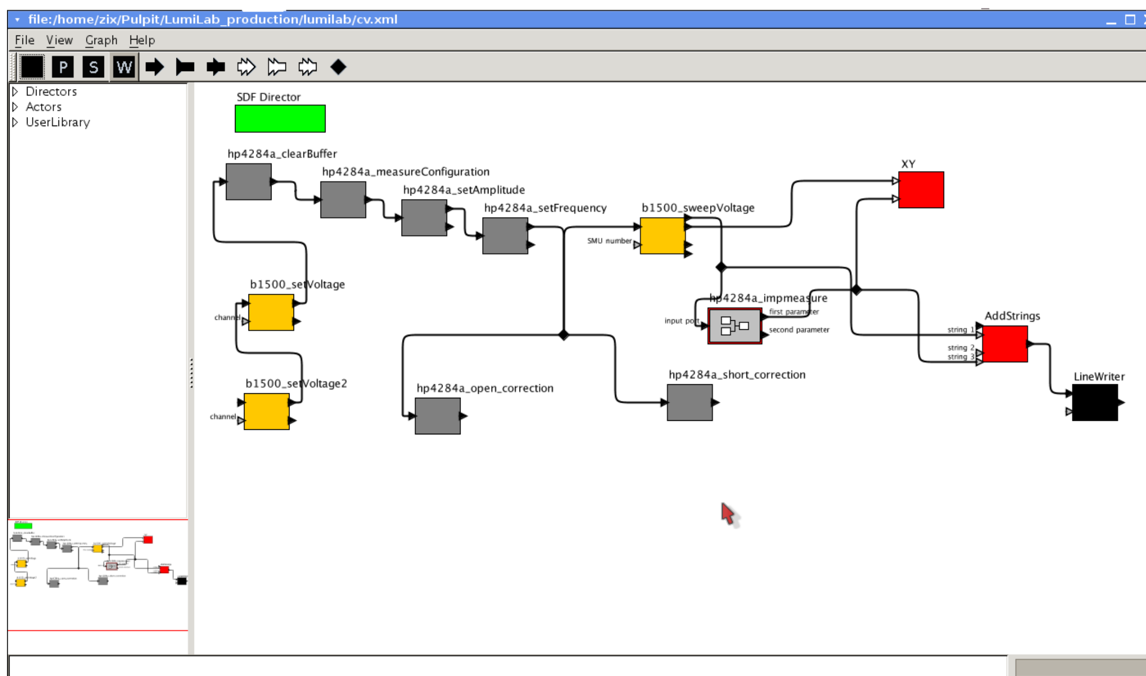
Pomiary prowadzone były za pomocą miernika LRC HP4284A i analizatora B1500A. Urządzenie B1500A wykorzystywane było do polaryzacji(podawania napięcia), natomiast miernikiem LRC mierzono składowe impedancji.

Na rysunku 5.14 zaprezentowano ogólny schemat stanowiska pomiarowego.



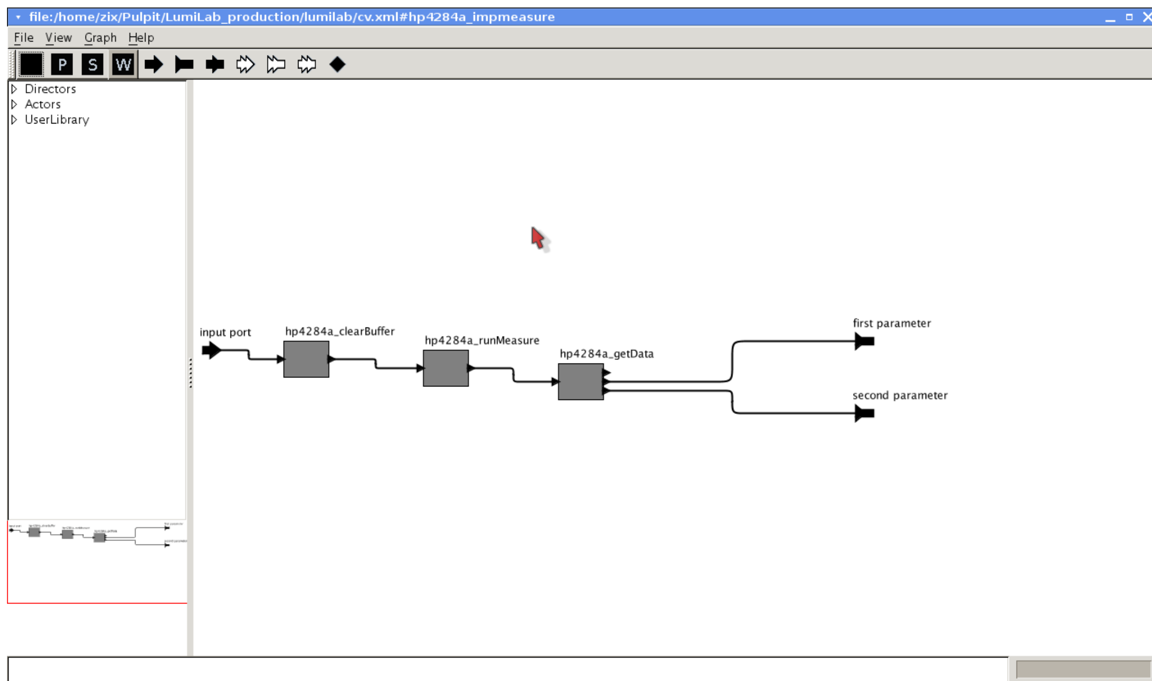
Rysunek 5.14. *Schemat stanowiska pomiarowego do pomiarów C-V sensorów.*

Stworzony model jest już bardziej skomplikowany z racji wykorzystania dwóch urządzeń i specyficznych funkcji jakie obsługuje miernik HP4284A. Na rysunku 5.15 zaprezentowano wspomniany model.



Rysunek 5.15. *Model dla pomiarów C-V.*

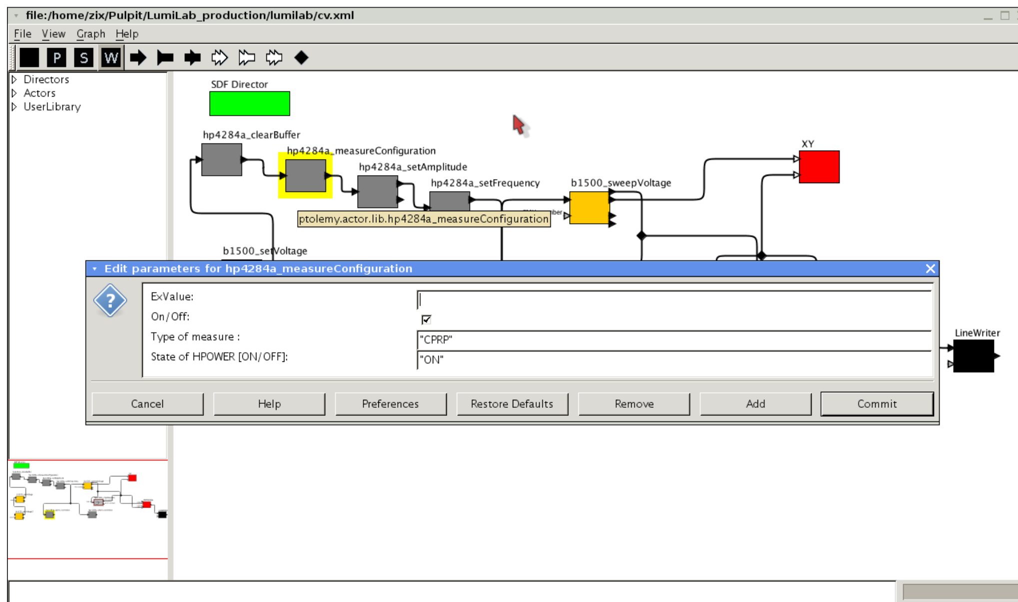
Jak widać model wykorzystuje aktora pochodnego(„hp4284A-impmeasure”). Jego wnętrze zaprezentowano na rysunku 5.16.



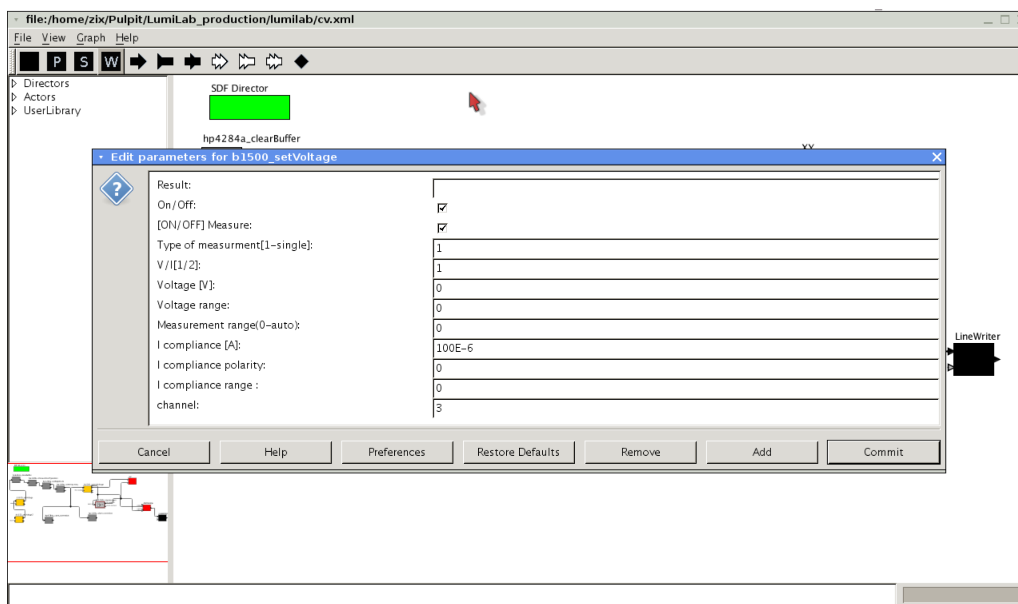
Rysunek 5.16. *Budowa aktora pochodnego wykorzystywanego w modelu do pomiarów C/V.*

Zadaniem tego aktora jest wykonanie pomiaru i przekazanie zebranych danych do komputera PC celem dalszej obróbki.

W modelu widać dwóch aktorów o nazwach „hp4284a-open-correction” i „hp4284a-short -correction”, ich zadaniem jest przeprowadzanie korekcji z sondami pomiarowymi rozwartymi i zwartymi(korekcje tą przeprowadza się aby wwyeliminować wpływ pojemności sond na pomiar). W opcjach aktorów zaznaczyć można, czy aktor ma pracować (byc aktywny), czy ma zostać zdeaktywowany. A więc zanim użytkownik zacznie pomiar, należy włączyć kolejno owych aktorów i przeprowadzić korekcje (uruchamiając model). Następnie należy ich zdeaktywować, aby można było wykonać właściwe pomiary. Z oczywistych powodów procesu korekcji nie da się w pełni zautomatyzować. Aktorzy ci posiadają rozbudowane menu graficzne wykorzystujące sbibliotekę swing z języka Java. Jest to dobry przykład pokazujący jak duży wpływ na wygląd aktora ma użytkownik. Dzięki zastosowaniu takich ułatwień proces korekcji jest bardzo mocno zautomatyzowany. Opcje do nastawiania przez użytkownika posiadają także aktorzy: „hp4284a -measureConfiguration”, „b1500 -setVoltage”, „b1500 -sweepVoltage”. Opcje dla pierwszego przedstawiono na rysunku 5.17. Opcje możliwe do nastawiania to typ pomiaru (na widocznym obrazku ustawiono CP-RP, czyli pomiar pojemności i rezystancji równolegle), a także włączenie i wyłączenie modu pracy „High power”, musi on być włączony jeżeli urządzenie ma być polaryzowane z zewnątrz (a w prezentowanym modelu tak jest). Dodatkowo widoczna jest wspomniana opcja, która umożliwia dezaktywowanie aktora (dazaktywacja oznacza, iż w danym modelu aktor jest „przezroczysty”, to znaczy nie są wykonywane żadne jego funkcje). Opcje dla aktorów „b1500 - setVoltage”, „b1500-sweepVoltage” są przedstawione na rysunkach 5.18 i 5.19.



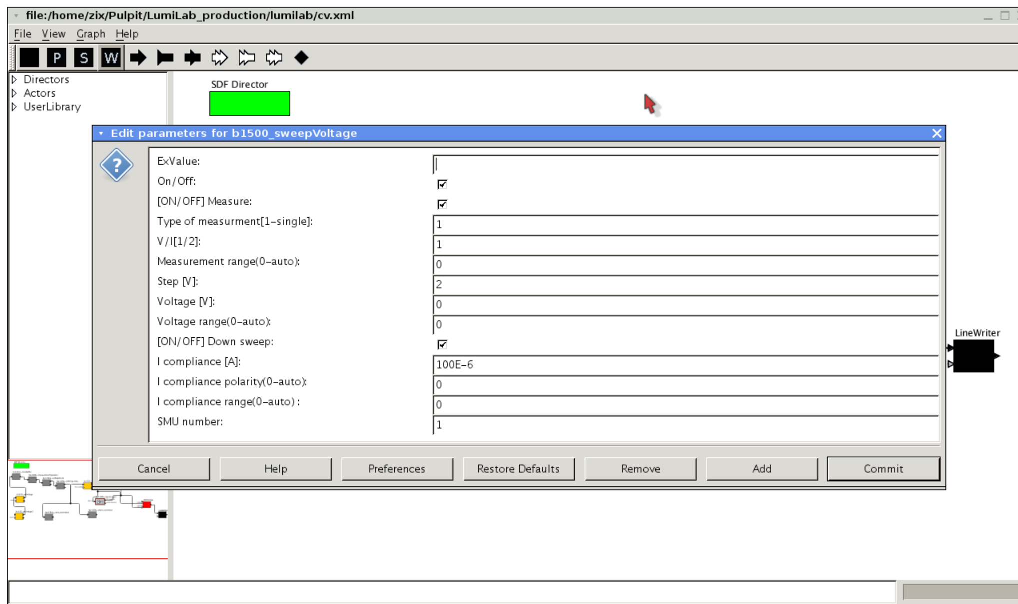
Rysunek 5.17. *Opcje dla aktora „hp4284a-measureConfiguration”.*



Rysunek 5.18. *Opcje dla aktora „b1500-setVoltage”.*

Są to typowe ustawienia, których znaczenia nie trudno rozszyfować. Pozwalają one na ustawienie opcji dla wybranego urządzenia.

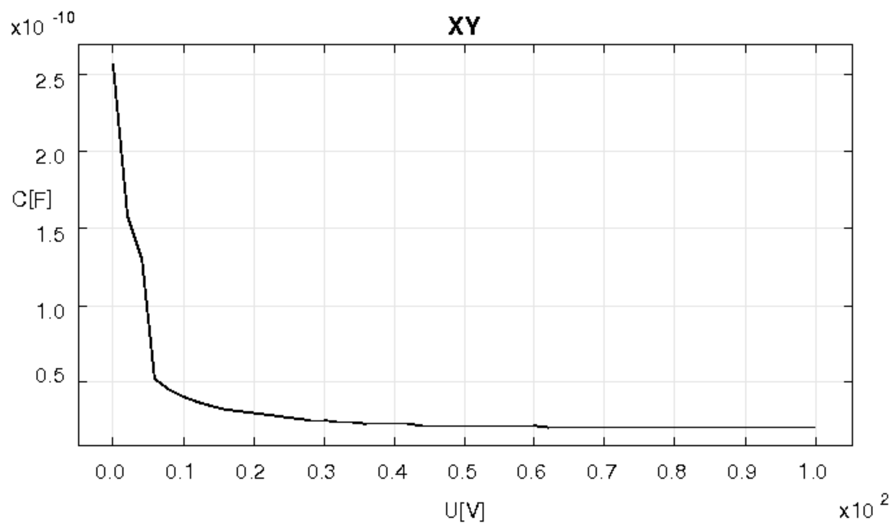
Wyniki zapisywane są do pliku o nazwie wskazanej w aktorze „LineWriter”, celem umożliwienia ich dalszej obróbki.



Rysunek 5.19. *Opcje dla aktora „b1500-sweepVoltage”.*

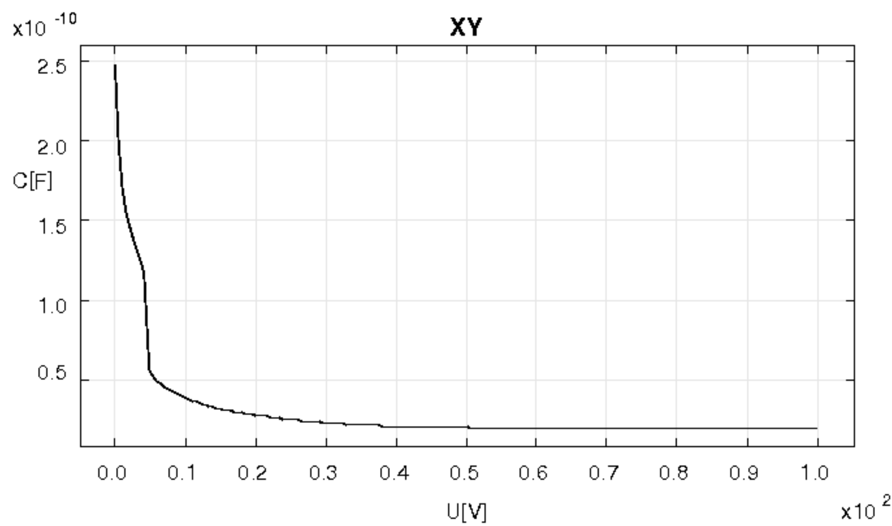
5.2.4. Wyniki

Wyniki zebrane zostały, podobnie jak w pomiarach I-V, dla dwóch padów detektora o numerach 63 i 64. Wyniki bardzo dobrze zgadzają się w tymi podanymi przez producenta, przedstawiono je na rysunkach 5.20 i 5.21.



Rysunek 5.20. *Wyniki pomiarów C-V dla pad numer 63.*

Dzięki zbudowaniu modelu w środowisku GUI pomiary zostały znacząco przyspieszone. Cały proces został w pełni zautomatyzowany (oprócz procesu korekcji, który jednakże wykonywany jest tylko jeden raz na początku pomiarów), a ingerencja użytkownika potrzeba jest jedynie, gdy trzeba zmienić konfigurację sprzętową badanego układu. Tak jak wspomniano wyniki bardzo dobrze zgadzają się z tymi podanymi przez producenta. Plik wyjściowy w łatwy sposób pozwala na przeprowadzenie dalszej obróbki danych. Udało się także dodać opcje, która umożliwia wyświetlanie wyników jako $\frac{1}{C^2}$ - V, co ułatwia wyznaczenie napięcia pełnego

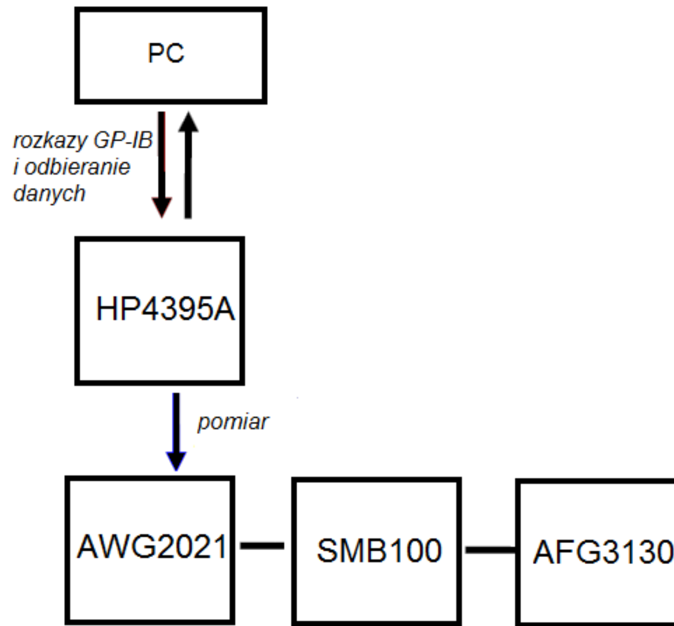


Rysunek 5.21. *Wyniki pomiarów C-V dla pad numer 64.*

zubożenia. Tak jak dla pomiarw I-V wyniki okazały się na tyle zadowalające, iż wykorzystane były w materiałach do prezentacji dla kolaboracji FCAL.

5.3. Badanie generatorów w celu określenia ich parametrów

Celem tego stanowiska pomiarowego było zbadanie parametrów widmowych trzech dostępnych generatorów tj. AWG2021, AFG3130 i SBM100. Określono jakość generowanego przebiegu sinusoidalnego względem generowanych przez urządzenie szumów i harmonicznych. Pomiary te miały wskazać, który generator najlepiej nadaje się do zastosowania w pomiarach zaprojektowanego w zespole Elektroniki Jądrowej 10 bitowego układu ADC [34]. Z racji dokładności, wymagana jest duża czystość podawanych na wejście ADC przebiegów sinusoidalnych. Pomiary miały wskazać, który generator podaje najczystszy przebieg, i ewentualnie, które zniekształcenia należy odrzucić, jako te pochodzące od generatora. Stanowisko składało sie jedynie z analizatora widma HP4395A i badanych generatorów. Zaprezentowano je na rysunku 5.22.



Rysunek 5.22. *Schemat stanowiska pomiarowego do badania generatorów.*

5.3.1. Procedura Pomiarowa

Pomiary prowadzone były na analizatorze widma HP4395A. W celu automatyzacji pomiaru napisany został odpowiedni program w języku C++, a następnie podpięty został do stworzonego środowiska graficznego. Program obejmuje nastawianie parametrów, takich jak szerokość przemiatanego pasma, częstotliwość środkową i szerokość pasma całkowitego. Ustalono, iż wyniki zebrane zostaną dla częstotliwości 1MHz i 10MHz, przy trzech różnych amplitudach sygnału 0.2V, 1V i 2V. Pomiary prowadzono w modzie „Noise" na kanale pomiarowym R (urządzenie posiada trzy kanały R,A,B, w modzie „Noise" korzysta się z tego pierwszego, natomiast w modzie „Network", port R służy do podawania sygnału, a A,B do mierzenia wartości), z tłumieniem ustawionym na 20dB(z wyjątkami o których wspomniano w dalszej części rozdziału).

Napisane oprogramowanie wyposażone jest w funkcje do liczenia odpowiednich parametrów. Wyniki opracowywane były zgodnie z informacjami zawartymi w artykule [35], który standaryzuje procedury pomiarowe dla układów ADC. Z tych względów mierzone były dwa parametry:

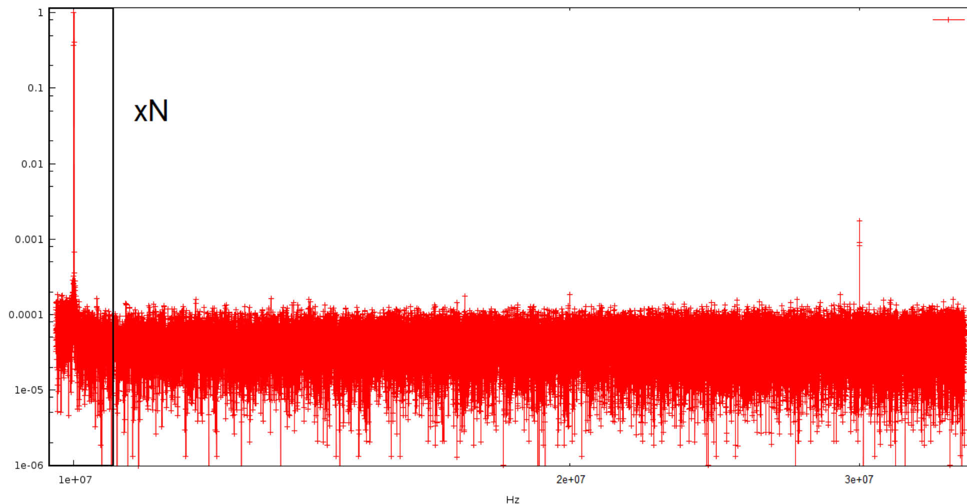
- SINAD (Signal-to-noise-and-harmonic-disortion) - według podanego artykułu jest to stosunek sygnału sinusoidalnego, do sygnału pochodzącego od szumów i harmonicznych.

- SNHR (Signal-Non-Harmonic-Ratio)- Jest to stosunek sygnału do szumu bez uwzględniania harmonicznych

Ze względu na podobieństwa pomiędzy SNR, a SNHR w procedurach pomiarowych układów ADC częściej stosuje się pojęcie SNHR. Tak też czyniono.

Przechodząc do opisu funkcji zawartych w napisanym oprogramowaniu wspomnieć należy, iż analizator posiada 801 komórek wewnętrznych pamięci, z których można odczytać informacje.

Aby więc uzyskać lepszą dokładność należy pomiar prowadzić w wąskich pasmach i wynik za pomocą oprogramowania scalać do oczekiwanego pasma całkowitego. Sposób podziału pasma całkowitego na podpasma pokazano na rysunku 5.23.

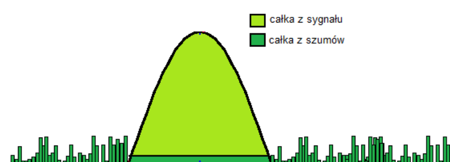


Rysunek 5.23. *Podział pasma całkowitego na N podpasm*

Pasmo główne jest dzielone na N części. W każdej z tych części prowadzony jest pomiar, a dane przesyłane są do komputera, który na końcu scala wszystkie części w całość. Dzięki temu można prowadzić pomiary w szerokim zakresie nie tracąc dokładności, co przy wąskim piku sygnału (względem szerokości całego pasma) generowanego przebiegu sinusoidalnego, jest bardzo ważne.

Kolejną funkcją zawartą w programie jest liczenie średniej kwadratowej (RMS) z widma szumów, służące do określenia średniej amplitudy szumów. Aby tego dokonać program wycina piki pochodzące od sygnału, a także piki od kolejnych harmonicznych (do trzeciego rzędu włącznie), zastępując te części pasma średnią wartością amplitudy z punktów przylegających do danego piku. Średnia kwadratowa łącznie z amplitudą sygnału jest potrzebna do określenia parametru SNHR.

Aby wyznaczyć SINAD program korzysta z funkcji do obliczania mocy sygnału i szumów. Moc całkowita liczona jest metodą trapezów, jako numerycznie wyznaczana całka z obszaru w wybranym paśmie. Podobnie obliczana jest SNHR z uwzględnieniem wspomnianych założeń. Zobrazowano to na rysunku 5.24.



Rysunek 5.24. *Całka z sygnału i szumów. Sposób obliczania.*

Na podstawie powyższych danych liczony jest stosunek sygnału do szumu, czyli SNHR, a także SINAD.

1. SNHR obliczane jako stosunek amplitud

$$SNHR = 20 * log(\frac{P_{sygnalu}}{P_{szumw}})$$

2. SINAD obliczany jest jako stosunek wyznaczonych wartości skutecznych

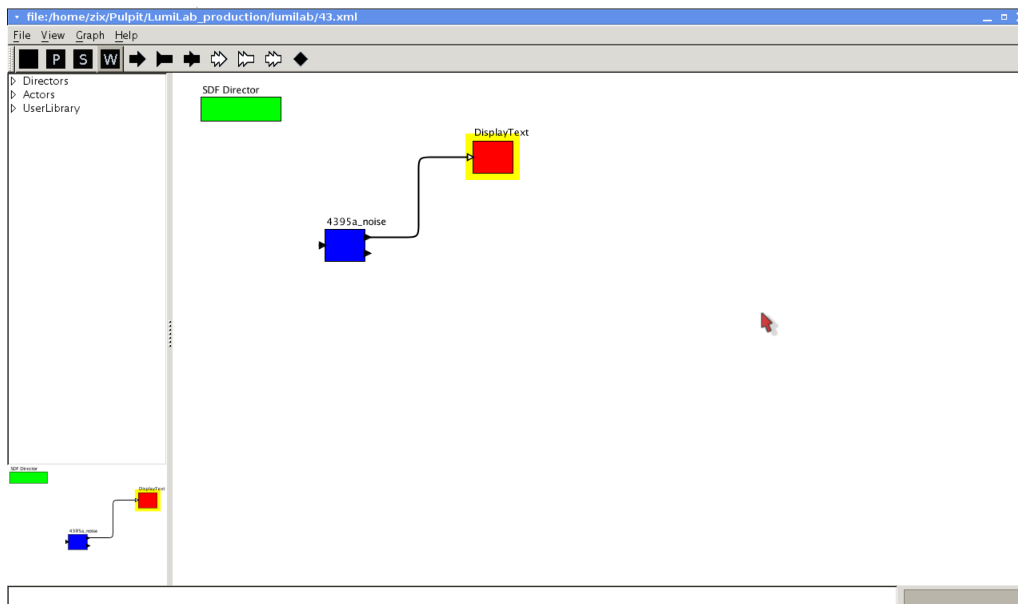
$$SINAD = 20 * log(\frac{P_{sygnalu}}{P_{szumwiharmonicznych}})$$

Amplituda sygnału wyznaczana jest jako maksimum z tablicy przechowującej dane. Logarytmy liczone są aby zamienić wartości z [V] na wartości w [dB].

Na koniec wspomnieć należy, iż dane zapisywane są w plikach tekstowych, co umożliwia ich dalszą obróbkę w programach zewnętrznych (na przykład takich jak GNUPLOT).

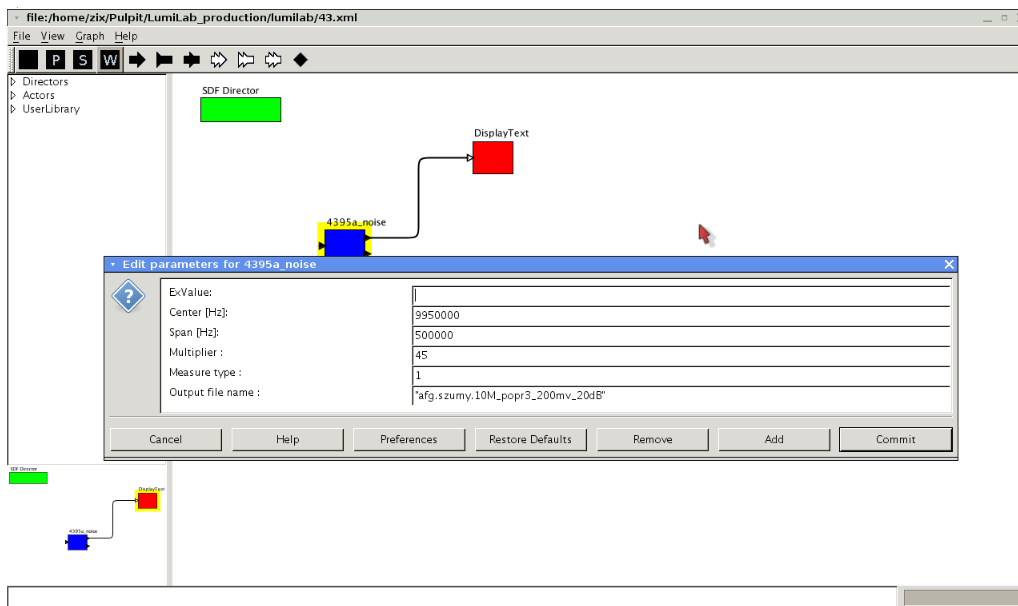
5.3.2. Model wykorzystywany w pomiarach

Model w tym przypadku jest bardzo prosty składa się zaledwie z dwóch aktorów. Wszystkie funkcje zawarte są wewnątrz aktora „noise”. Wygląd modelu przedstawiono na rysunku 5.25.



Rysunek 5.25. *Model do pomiarów parametrów generatorów.*

Aktor „noise” pozwala na nastawienie kilku parametrów. Ustawić można szerokość pasma podstawowego, a także mnożnik który powoduje rozszerzenie tego pasma (algorytm opisany jest w podrozdziale wyżej). Dodatkowo ustawić można typ pomiaru (SPECTRUM/NOISE(1/2)), a także nazwę zbioru do jakiego zapisane zostaną wyniki. Parametry aktora przedstawione są na rysunku 5.26.



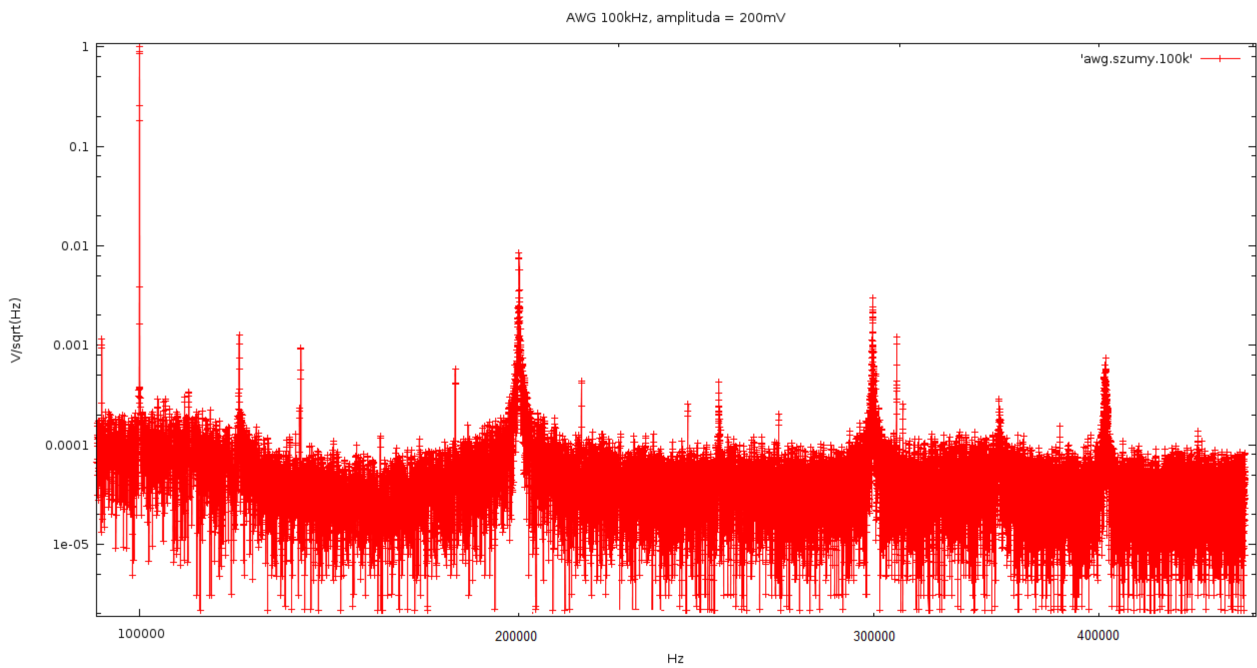
Rysunek 5.26. *Opcje dla aktora „Noise”.*

Aktor „DisplayText” umieszczony jest, aby użytkownik wiedział kiedy skończył się pomiar. Przekazywane są do niego odpowiednie komunikaty, które wyświetlane są po zakonczeniu działania modelu.

5.3.3. Pomiary dla generatora AWG2021

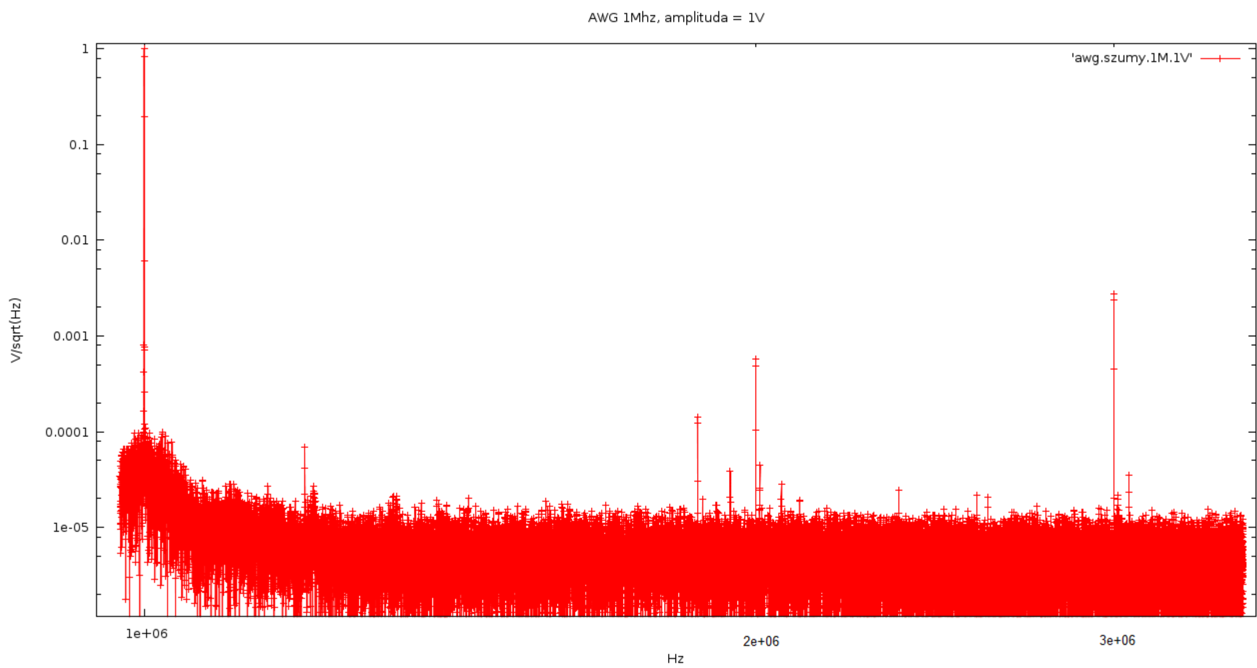
AWG2021 jest generatorem firmy Sony/Tektronix. Maksymalna częstotliwość generowanego przebiegu to 2.5 MHz, maksymalna amplituda sygnału dla tego urządzenia to 5V. A więc niemożliwym było sprawdzenie dla wybranych częstotliwości 1MHz i 10MHz, dlatego też przeprowadzono pomiary przy 100 kHz i 1 MHz. Pomiary dla 1MHz przeprowadzone zostały dla trzech wskazanych amplitud sygnału 0.2V, 1V i 2V.

Na rysunkach 5.27,5.28,5.29,5.30 umieszczone zostały zebrane widma. Każde wykreślone zostało w skali Log/Log i znormalizowane w celu lepszego zobrazowania różnic wobec innych badanych generatorów.

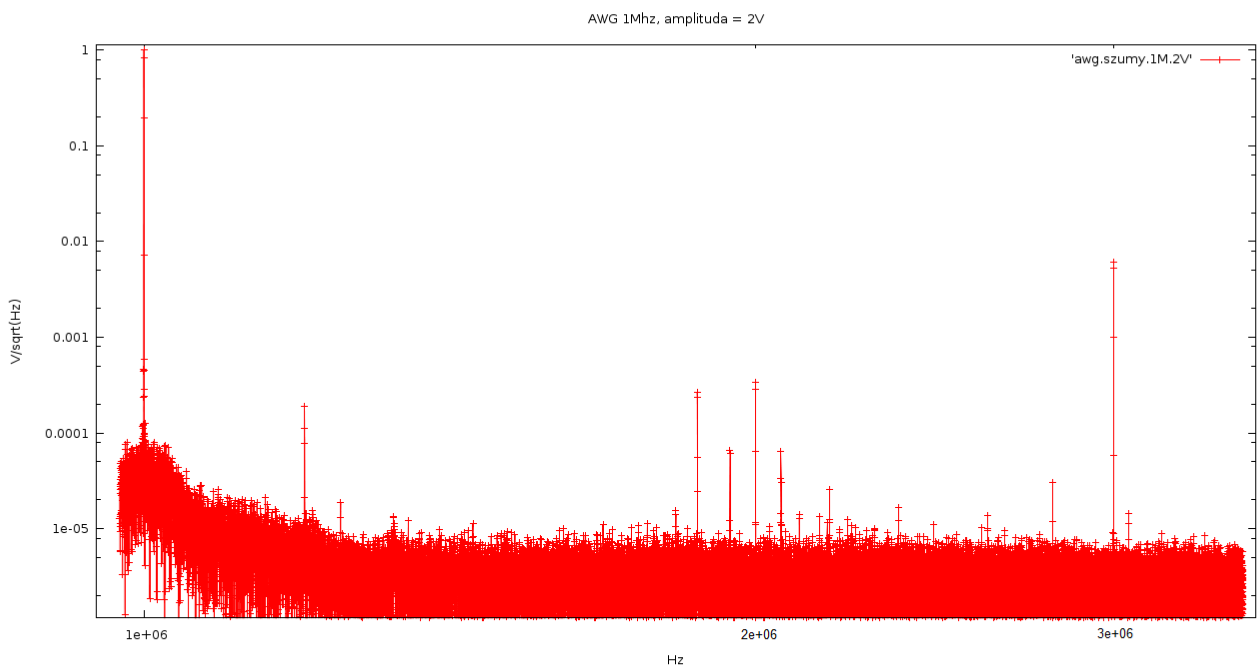


Rysunek 5.27. *Widmo dla generatora AWG2021 prz amplitudzie 0.2V i f=100kHz.*

Rysunek 5.28. *Widmo dla generatora AWG2021 przy amplitudzie 0.2V i f=1MHz.*



Rysunek 5.29. *Widmo dla generatora AWG2021 przy amplitudzie 1V i f=1Mhz.*



Rysunek 5.30. *Widmo dla generatora AWG2021 przy amplitudzie 2V i f=1MHz.*

Zaobserwować można wzrost jakości sygnału wraz ze wzrostem amplitudy sygnału. Urządzenie generuje sygnał czysty, a występujące harmoniczne mają znaczenie marginalne. Niestety ograniczeniem jest wąskie pasmo, sięgające jedynie 2.5 MHz. Dla niższych częstotliwość urządzenie wydaje się bardzo dobrym jako składowa systemów pomiarowych dla badanego układu ADC. Dla 100kHz da się zauważyć realtywnie wysokie szumy względem sygnału, natomiast dla 1MHz ciekawym jest fakt, iż trzecia harmoniczna jest znacznie większa niż druga. Wyniki pomiarowe zebrane są w tabeli 5.2.

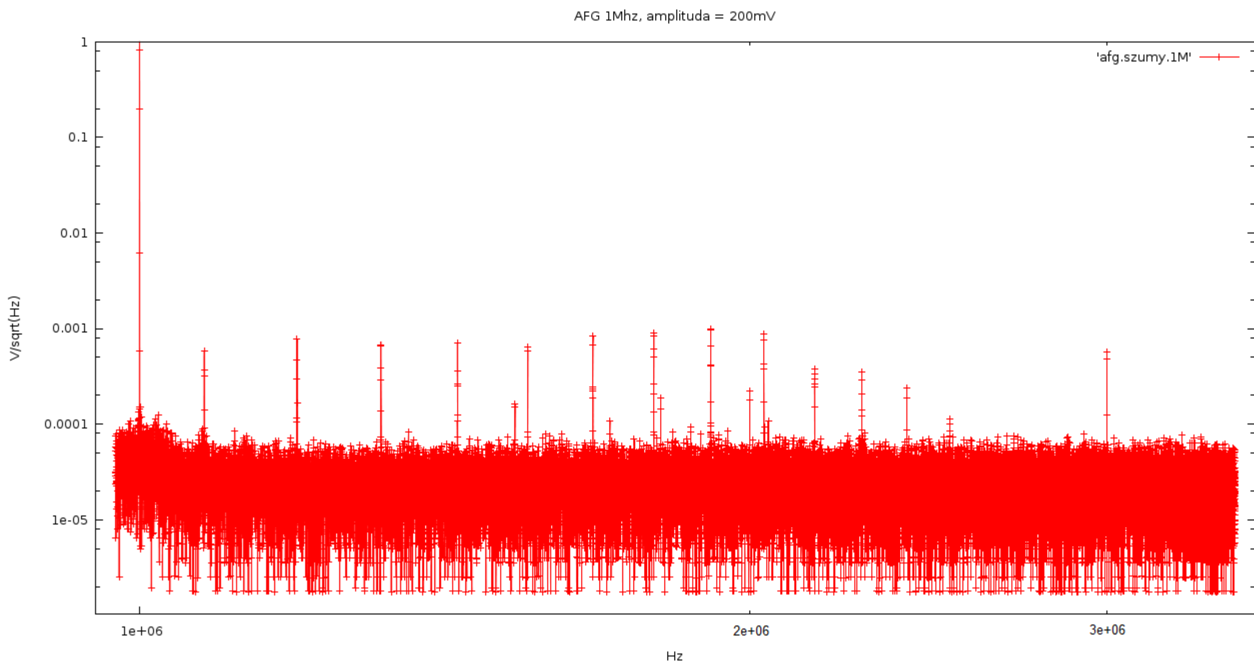
	100kHz, 0.2V	1Mhz, 0.2V	1Mhz, 1V	1Mhz, 2V
SNHR [dB]	76	91	100	101
SINAD [dB]	56	47.5	57	57.5

Tablica 5.2. *Obliczenia wartości SNHR i SINAD dla generatora AWG2021.*

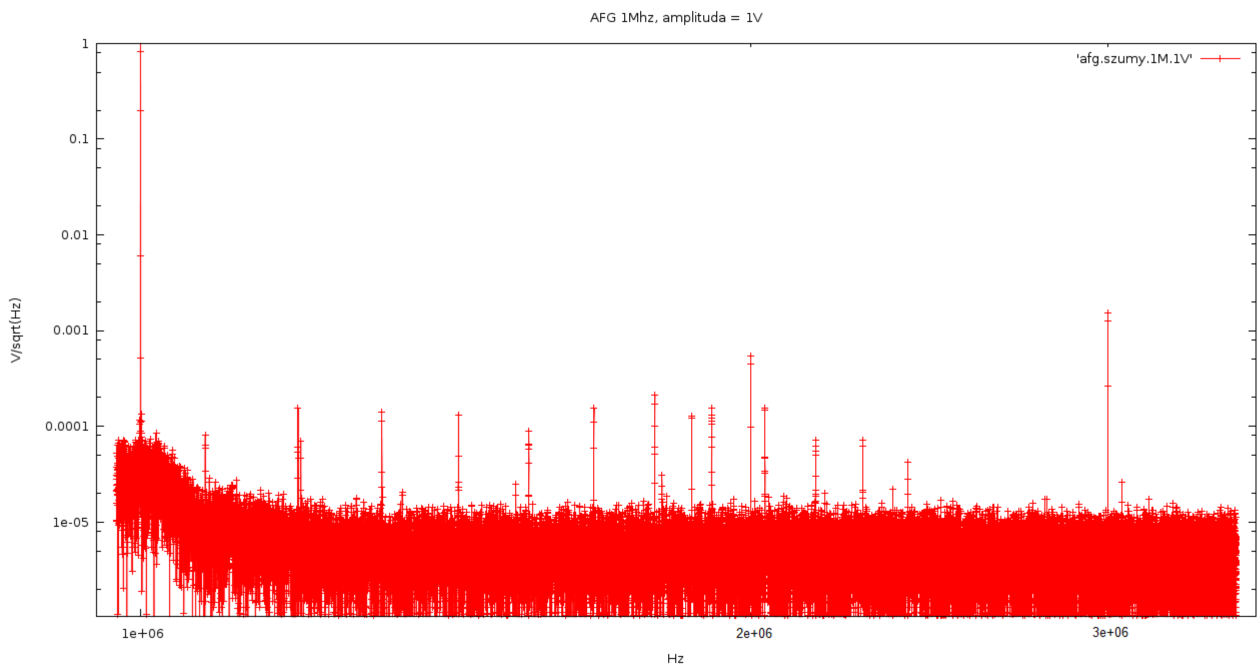
5.3.4. Pomiary dla generatora AFG3102

Generator AFG3102 jest „młodszym bratem„ generatora AWG2021. Posiada podobne parametry, ale pasmo zostało zwiększone do 100MHz. Dla tego generatora zebrano widma dla częstotliwości 1 MHz i 10 MHz, przy wskazanych trzech amplitudach.

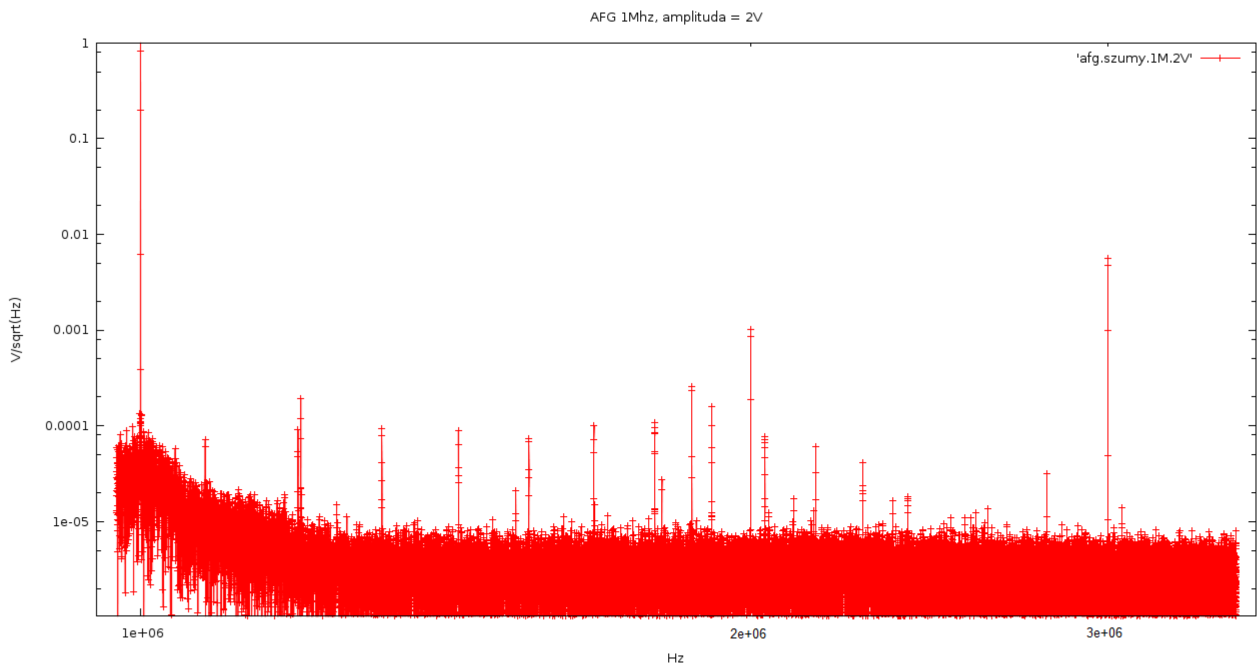
Na rysunkach 5.31,5.32,5.33 przedstawiono uzyskane widma dla częstotliwości 1MHz. Tak jak poprzednio wszystkie znormalizowane i w skali Log/Log.



Rysunek 5.31. *Widmo dla generatora AFG3102 przy amplitudzie 0.2V i f=1MHz.*

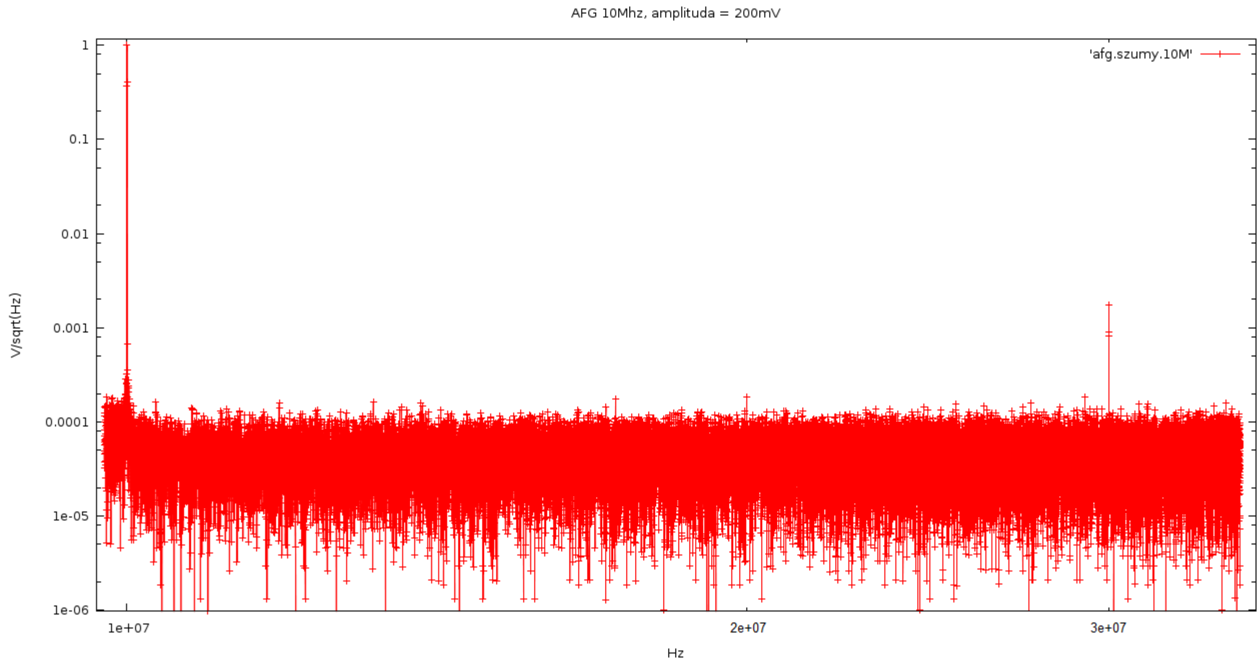


Rysunek 5.32. *Widmo dla generatora AFG3102 przy amplitudzie 1V i f=1MHz.*

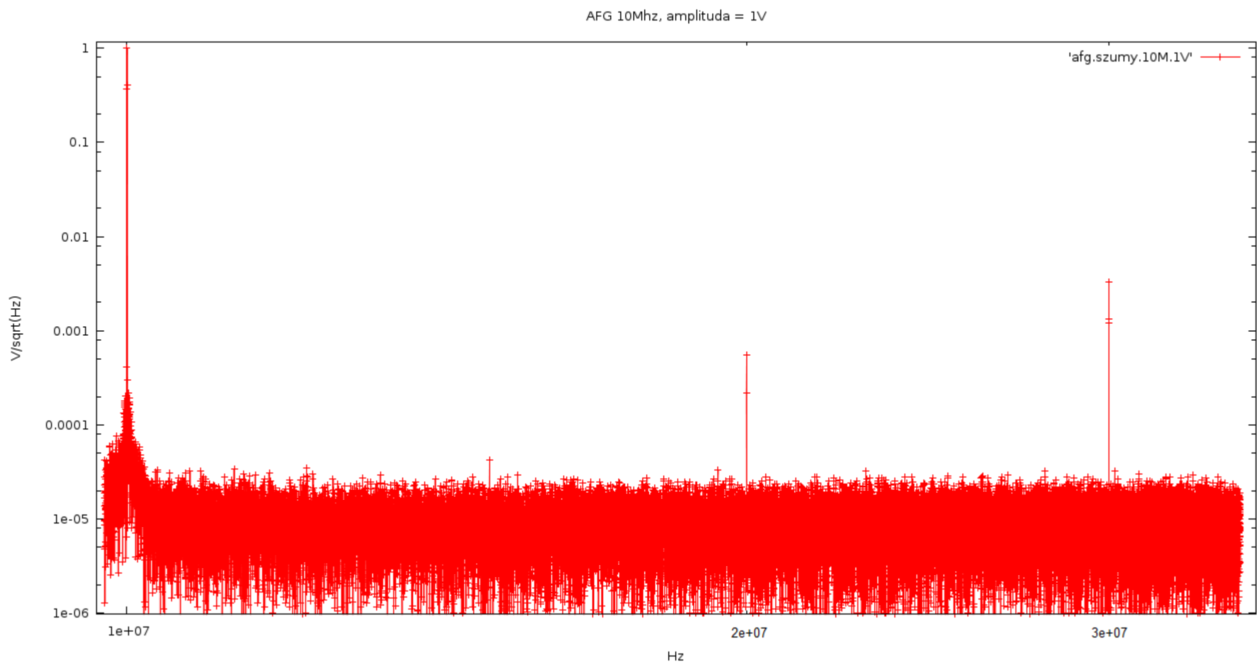


Rysunek 5.33. *Widmo dla generatora AFG3102 przy amplitudzie 2V i f=1Mhz.*

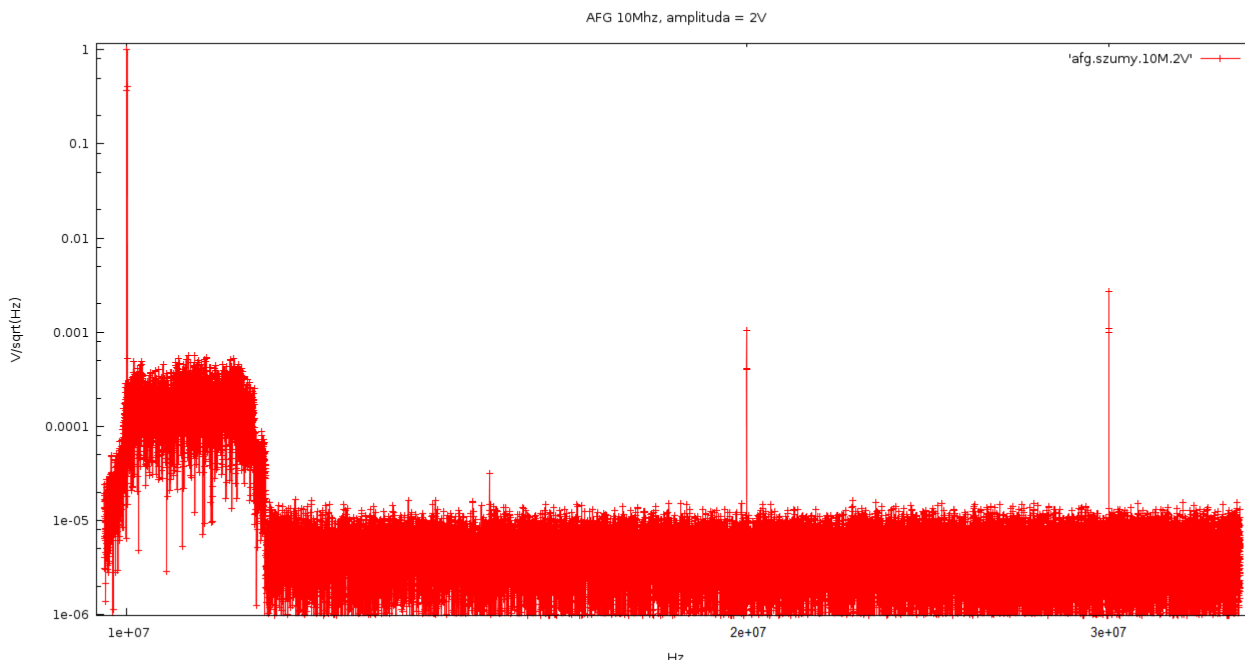
Widma dla częstotliwośći 10 MHz, przedstawiono na rysunkach 5.34,5.35,5.36.



Rysunek 5.34. *Widmo dla generatora AFG3130 przy amplitudzie 0.2V i f=10MHz.*



Rysunek 5.35. *Widmo dla generatora AFG3130 przy amplitudzie 1V i f=10MHz.*



Rysunek 5.36. *Widmo dla generatora AFG3130 przy amplitudzie 2V i f=10MHz.*

Na widmach dla f=1MHz da się zauważyć znaczną liczbę „wąsów” w okolicy tej częstotliwości. Jest to najpewniej cecha badanego generatora. Dla pewności wykonano pomiary dla częstotliwości bliskich tak jak 2 MHz i 3 MHz, na widmach tych da się zauważyć, iż te większe piki występują jedynie w okolicy 1 MHz. Potwierdza to, iż zakłócenia te pochodzą wprost z badanego generatora, nie występują one bowiem dla żadnej badanej wyższej częstotliwości. Kolejnego komentarza wymaga także widmo dla 10MHz i amplitudy 2V, gdzie w okolicach piku widać znaczny „garb”. Ustalono, iż jest on spowodowany niewystarczającym tłumieniem na wejściu R analizatora pomiarowego. Przy ustawieniu tłumienia na 40 dB, garb zmniejsza się, ale większe tłumienie podnosi całkowity próg szumów. Garbu tego nie uwzględniano w obliczeniach. Druga i trzecia harmoniczna są wiekszę niż dla generatora AWG2021. Także i tutaj trzecia harmoniczna jest większa niż druga.

Tak jak dla AWG2021, obliczono SNHR i SINAD. Wyniki prezentowane są w tabeli 5.3.

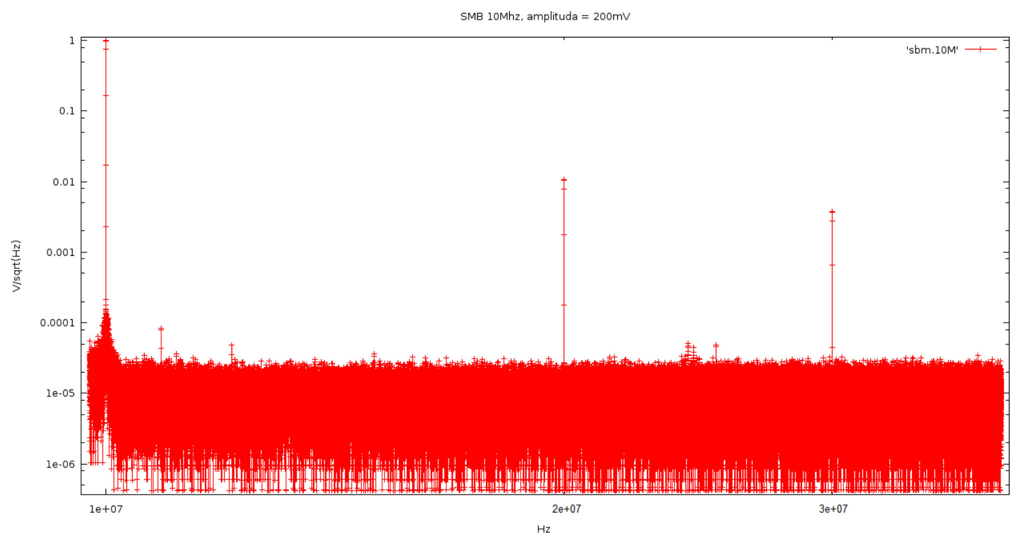
	1MHz, 0.2V	1Mhz, 1V	1Mhz, 2V	10Mhz, 0.2V	10MHz, 1V	10MHz, 2V
SNHR [dB]	88	101	102	86	98	86
SINAD [dB]	46	57	59	41	53.5	41.5

Tablica 5.3. *Obliczenia wartości SNHR i SINAD dla generatora AFG3031.*

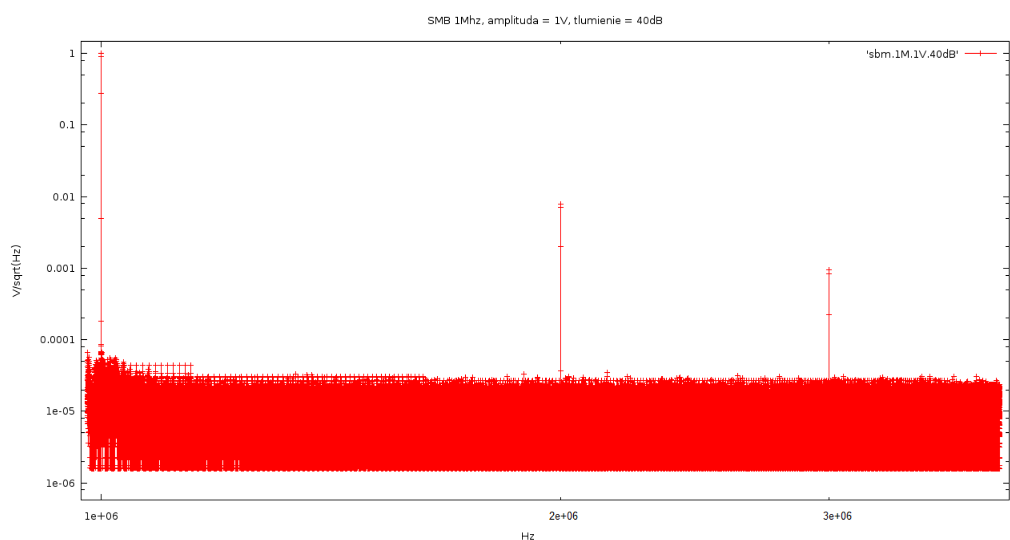
5.3.5. Pomiary dla generatora SMB100

Ostatnim badanym generatorem był generator SMB100, firmy RiS. Posiada on najszersze pasmo, dochodzące do 3 GHz, a maksymalna amplituda sygnału to 7V. Urządzenie badane było dla dwóch wskazanych częstotliwości i trzech amplitud.

Na rysunkach 5.37,5.38 przedstawiono widma dla częstotliwości 1 MHZ. Tak jak poprzednio wszystkie zostały znormalizowane i przedstawione w skali Log/Log.

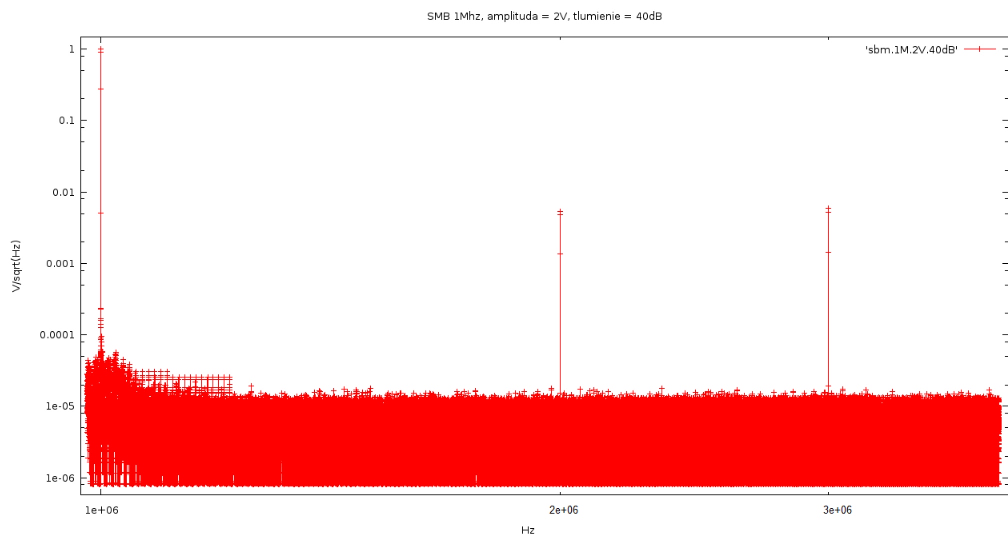


Rysunek 5.37. *Widmo dla generatora SMB100 przy amplitudzie 0.2V i f=1MHz.*



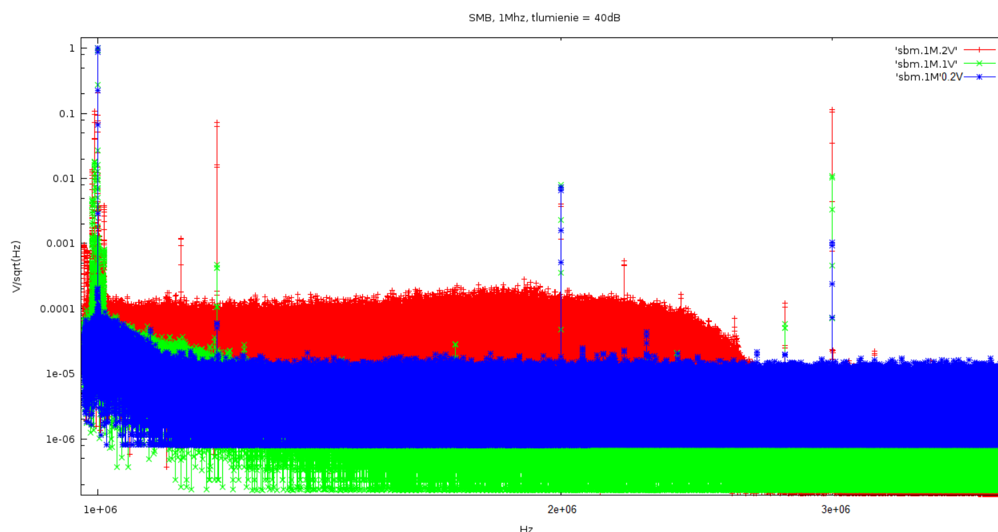
Rysunek 5.38. *Widmo dla generatora SMB100 przy amplitudzie 1V i f=1MHz.*

Jak widać, już przy sygnale o amplitudzie 1V analizator ma problemy z tłumieniem. Najprościej było by zwiększyć tłumienie do odpowiedniej wartości. Autor zbadał jednak, iż krok taki podnosi lekko ogólny poziom szumów, co uniemożliwiało by miarodajne porównywanie wyników. Na rysunku 5.39 zaprezentowano wyniki dla amplitudy 2V.



Rysunek 5.39. *Widmo dla generatora SMB100 przy amplitudzie 2V i f=1MHz.*

Podniesienie tłumienia niweluje garb, ale jak było już wspomniane sztucznie podnoszony jest poziom szumów. Co więcej stwierdzono, iż nawet ustawienie maksymalnej wartości tłumienia na wejściu R, to jest 40dB, nie jest wystarczające. Zobrazowano to na zbiorczym widmie przedstawionym na rysunku 5.40. Jest to widmo dla wszystkich trzech amplitud, f=1MHz, i tłumienia 40dB.

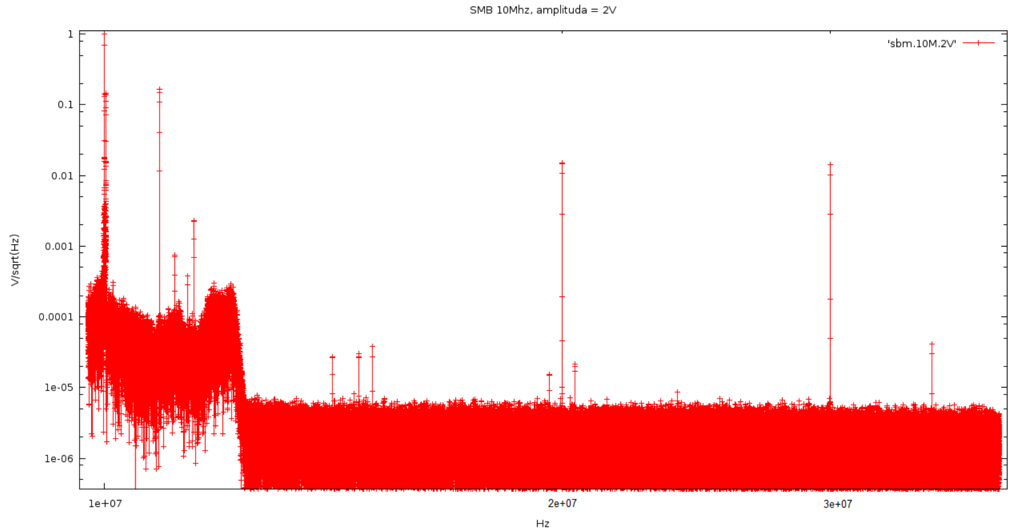


Rysunek 5.40. *Widmo zbiorczegeneratora SMB100 i f=1MHz.*

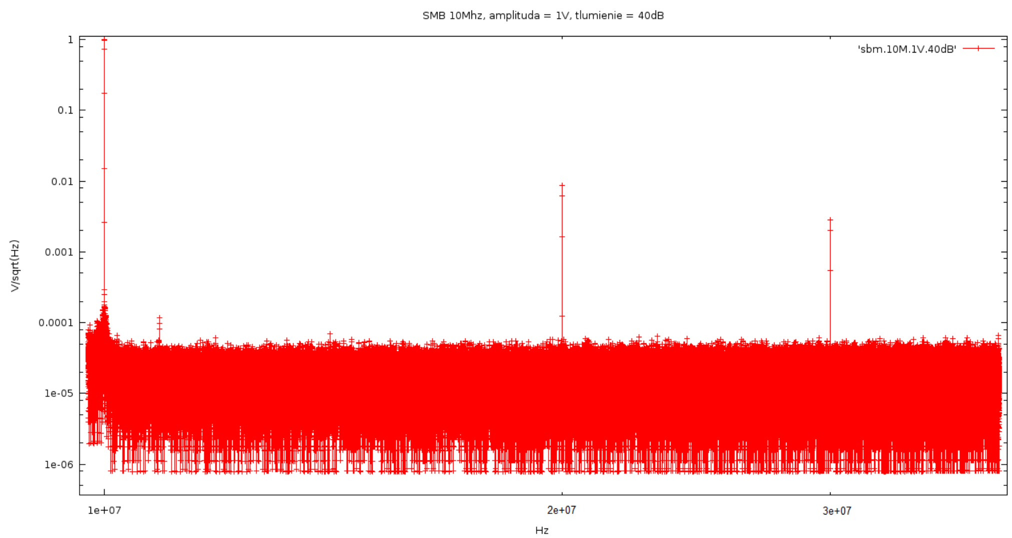
Widać wyraźnie, iż nadal istnieją zakłócenia spowodowane przeładowaniem wejścia analizatora HP4395A, a dodatkowo zwiększone zostały szumy.

Aby rozwiązać ten problem, postanowiono nie uwzględniać tej zniekształconej części widma w obliczeniach parametrów. Uzyskane w ten sposób wyniki wydają się dobre.

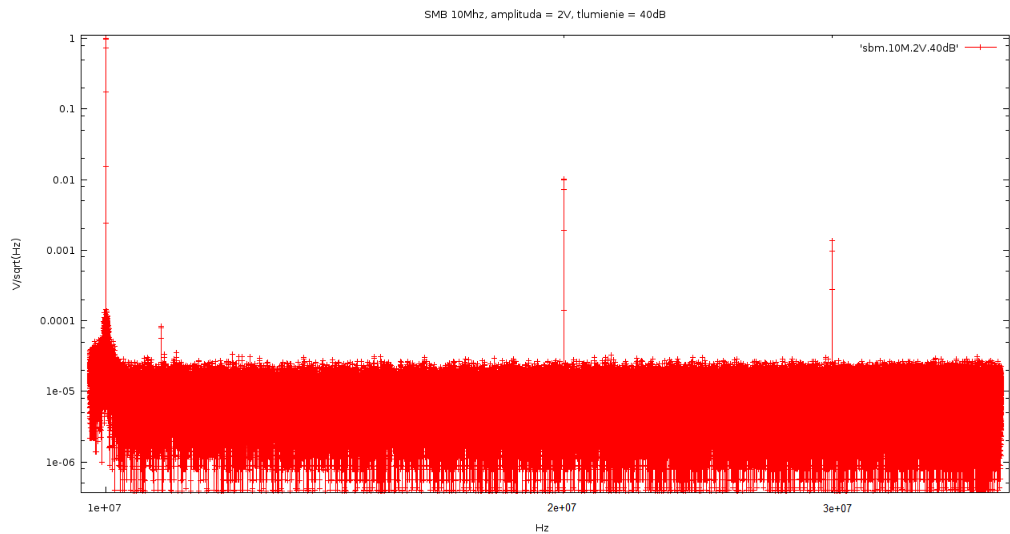
Na rysunkach 5.41,5.42,5.43 umieszczono widma dla częstotliwości 10MHz.



Rysunek 5.41. *Widmo dla generatora SMB100 przy amplitudzie 0.2V i f=10MHz.*



Rysunek 5.42. *Widmo dla generatora SMB100 przy amplitudzie 1V i f=10MHz.*



Rysunek 5.43. *Widmo dla generatora SMB100 przy amplitudzie 2V i f=1MHz.*

Tutaj także widać wspomniane zakłócenia pochodzące od niewystarczającego tłumienia, i tak jak poprzednio zastosowano podobną technikę aby nie brać ich pod uwagę w obliczeniach.

Dodatkowo tak jak dla innych analizatorów określono dwa wybrane parametry SNHR i SINAD. Wyniki zamieszczono w tabeli 5.4.

	1MHz, 0.2V	1Mhz, 1V	1Mhz, 2V	10Mhz, 0.2V	10MHz, 1V	10MHz, 2V
SNHR [dB]	88	101	102	86	98	86
SINAD [dB]	46	57	59	41	53.5	41.5

Tablica 5.4. *Obliczenia wartości SNHR i SINAD dla generatora SMB100.*

5.3.6. Wnioski

Po przeprowadzeniu wszystkich pomiarów i określeniu parametrów, wnioskować można iż najczystszy sygnał względem szumów bez wątpienia posiada generator SMB 100. Natomiast najlepszy stosunek sygnału użytecznego do jego harmonicznych posiada generator AWG2021, niestety posiada on mocno ograniczone pasmo sięgające jedynie 2.5 MHZ. Podobne parametry do AWG, posiada AFG, czyli jego młodszy brat. Stwierdzono jednak, iż w okolicy 1MHz, w generatorze tym występują zakłócenia, które podnoszą znacznie próg szumów.

Wniosek płynący z pomiarów, to próba zbudowania takiego stanowiska, w którym można by tak manipulować urządzeniami(poprzez na przykład oprogramowanie), aby sygnał w zakresie poza harmonicznymi pochodził od SMB100, a w okolicy harmonicznych od AFG3102. Takie stanowisko z pewnością zapewniało by najczystszy generowany sygnał.

5.4. Stanowisko do pomiarów szumów układu elektroniki Front-End

Badany układ elektroniki Front-End [36] podobnie jak inne badane układy, zaprojektowany został jako część elektroniki odczytowej przyszłego detektora LumiCal. Celem było zbudowanie stanowiska pomiarowego dookreślenie szumów na wyjściu układu w zależności od pojemności

wejściowej. Stanowisko pomiarowe składało się z analizatora widma HP4395A i komputera PC z napisanym odpowiednim oprogramowanie sterującym zintegrowanym ze środowiskiem graficznym.

Niestety podczas pomiarów pojawiły się problemy z ustabilizowaniem układu. Mimo wielokrotnych prób układu nie udało sie ustabilizowac na tyle, aby możliwe były miarodajne pomiary, które mozna by porównać z pomiarami wykonanymi za pomocą analizatora HP4195A. Pomiędzy wcześniejszymi pomiarami, a obecnymi upłynęło około 1 rok czasu. W międzyczasie do układu dolutowane zostały pojemności około 3.3pF, uniemożliwiło to na obiektywne porównanie wyników. Co więcej istaniała ogromna niestabilność zbieranych wyników (wachały się na przykład dla danej pojemności pomiędzy 4 a 12 pF). Dlatego też postanowiono nie dolączać ich do pracy.

5.4.1. Procedury pomiarowe dla zaprojektowanego stanowiska

Procedury pomiarowe były zbliżone do tych stosowanych w pomiarach parametrów generatorów. Za pomocą analizatora mierzone było widmo szumów na podstawie którego za pomocą numerycznego wyznaczania całki liczona była moc szumów. Wspomnieć tutaj należy, iż wcześniej określane były parametry tego układu za pomocą analizatora HP4195A, czyli starszej wersji HP4395A.

5.4.2. Model wykorzystywany w pomiarach

Model korzysta z tych samych aktorów, które wykorzystywane były w pomiarach widm generatorów gdyż badane były podobne parametry. Sama obróbka danych jest prostsza, ponieważ nie ma tutaj konieczności kontroli tłumienia. Wszystkie pomiary przeprowadzono z atenuacją wynoszącą 0dB.

Podsumowanie

Celem pracy było stworzenie oprogramowania sterującego urządzeniami pomiarowymi i automatyzacja wybranych pomiarów w labolatorium Zespołu Elektroniki i Detekcji Cząstek. Dodatkowo własną inwencją autora była chęć stworzenia interfejsu graficznego, którego zadaniem miało być ułatwienie obsługi wspomnianych urządzeń. Aby zrealizować te cele praca podzielona została na etapy. W etapie pierwszym określono założenia i technologie wykorzystywane podczas tworzenia oprogramowania. W etapie drugim zrealizowano założone zadania i sprawdzono czy wybrane rozwiązania odpowiadają stawianym potrzebom (poprzez przeprowadzenie próbnych pomiarów).

W pierwszym etapie na najważniejszy wkład pracy składają się:

- Przedstawienie różnych koncepcji oprogramowania z argumentacją „za” i „przeciw” dla każdej z nich oraz wybranie koncepcji finalnej, optymalnej według autora.
- Określenie architektury oprogramowania, tzn. umiejscowienie w środowisku wszystkich elementów (sterowniki, środowisko graficzne, dane pomiarowe, narzędzia do obliczeń etc.). Uwzględnienie w architekturze wszystkich wymagań tzn. dużej elastyczności, prostej konfiguracji i łatwej rozbudowy o nowe elementy.
- Opracowanie systemu, który umożliwia budowanie systemów pomiarowych dwutorowo, tak aby możliwym było budowanie oprogramowania konsolowego, lub poprzez stworzone środowisko graficzne.
- Wybranie narzędzi za pomocą których tworzone były programy. Sterowniki, z racji wybranej biblioteki do sterowania GP-IB, pisane były w języku C++. Natomiast jako podstawę dla środowiska graficznego wybrano narzędzie Ptolemeusz, co narzuciło konieczność stosowania języków Java i XML.

W drugim etapie, czyli praktycznej realizacji systemu, najważniejsze wykonane zadania to:

- Stworzenie szeregu sterowników przeznaczonych dla wybranych urządzeń pomiarowych. Sterowniki tworzono zgodnie z tym co zostało juz wspomnianie, w oparciu o bibliotekę libgpib. Językiem programowania był C++. Stworzono sterowniki dla kilkunastu urządzeń, takich jak analizatory widma, zasilacze, analizatory urządzeń półprzewodnikowych, oscyloskopy etc.
- Zrealizowanie pierwszej wersji środowiska graficznego, które pozwalało by na wykonanie próbnych pomiarów i budowę przykładowych stanowisk pomiarowych. Pomiary te miały dać odpowiedź na pytanie, czy rozwój oprogramowania podąża we właściwym kierunku.
- Rozwijanie oprogramowania wraz z poprawkami uwzględniającymi sugestie użytkowników. Wprowadzanie dodatkowych opcji, zmiany w wyglądzie środowiska, dodawanie nowych przycisków etc.
- Wykonanie testowych pomiarów, których celem było sprawdzenie słuszności wyboru wykorzystywanych technologii i poprawności napisanego oprogramwoania. Wykonano cztery

stanowiska pomiarowe, za pomocą których przeprowadzono pomiary wybranych układów elektroniki i detektorów. Zbadano przetwornik cyfrowo-analogowy, generatory sygnału do testowania przetwornika analogowo-cyfrowego, krzemowe sensory promieniowania i układ elektroniki front-end.

Ponieważ stworznie kompletnego środowiska graficznego, jest zadaniem bardzo ambitnym (nad rozwiązaniami komercyjnymi, takimi jak na przykład LabView, pracuje sztab ludzi przez wiele lat, a sam Ptolemeusz rozwijany jest od roku 1987), stworzony przez autora system należy traktować jako wersję rozwojową środowiska pomiarowego. Dlatego też ostatnią częścią pracy było nakreślenie kierunków przyszłego rozwoju oprogramowania według wizji autora. Najważniejsze z nich to:

— Dalsze testy oprogramowania i wprowadzanie zmian w samym interfejsie graficznym zgodnie z potrzebami i sugestiami użytkowników (na przykład umiejscowienie elementów na paskach etc.).
— Tworzenie nowych aktorów wraz z pojawianiem się nowych urządzeń. Rozbudowy wymaga także baza aktorów użytkowych takich jak na przykład obsługa plików, obróbka zmiennych, rysowanie wykresów etc.
— Rozbudowa opcji pomocy w środowisku. Tutaj zostały poczynione już pewne prace, ale całość należy dalej usprawniać zgodnie z sugestiami użytkowników, tak aby pomoc dostępna była na każdym etapie tworzenia modelu.
— Stworzenie automatycznego systemu aktualizacji programów sterujących. Pierwsze próby w tym kierunku zostały już poczynione, ale nie wszystko w tym aspekcie zależy od samego środowiska. Wymaga to ujednolicenia zasad co do trzymania samych programów sterujących w bazie.
— Dalsze prace nad modułem do pracy zdalnej. Wykonanie kompletnego systemu do kontroli kont i do zdalnego łączenia się ze środowiskiem etc.
— „Odchudzenie” środowiska. Po Ptolemeuszu zostało jeszcze dużo elementów, które z punktu widzenia GUI są niepotrzebne. Należy je systematycznie usuwać.

Podsumowując, należy stwierdzić, że wszystkie postawione cele pracy zostały osiągnięte. Wybrane technologie są, według autora, optymalne i pozwalają w prosty sposób tworzyć oprogramowanie pomiarowe. W pracy wybrano architekturę i wykonano wszystkie założone elementy oprogramowania, tak konsolowego jak i graficznego. Z sukcesem przetestowano stworzone oprogramowanie, wykonując testowe pomiary. Autor zaproponował także kierunki przyszłego rozwoju.

Dodatek A

Instalacja libgpib

W dodatku tym zaprezentowany jest sposób instalacji w systemie oprogramowania do obsługi GP-IB. Sposób ten tyczy się systemów z rodziny Linux.

Pierwszą czynnością powinno być zainstalowanie z poziomu *root'a* następujących bibliotek:

```
libgpib0, libgpib0-dev, libgpib-bin
```

Najprościej można to zrobić(wymaga zainstalowanego apt) wydając polecenie w konsoli:

```
apt-get install libgpib0 libgpib0-dev libgpib-bin
```

Następnie stworzyć trzeba prosty skrypt, który „startował" będzie kontroler GP-IB.

Ostatnią czynnością jest modyfikacja pliku *gpib.conf* znajdującego się w katalogu */etc/*. W dziale oznaczonym jako *interface*, w urządzeniu pierwszym(oznaczonym jako *minor = 0*), należy edytować pozycję $board_type$, aby wyglądała następująco:

```
board_type = "ni_usb_b"
```

Od tej pory, po poprawnym uruchomieniu skryptu kontroler nadaje się do pracy.

Dodatek B

Biblioteka Stringutils

Metody z klasy GpibDev.

```
/**
   * zwraca blad GPIB, z dodatkowym komunikatem tekstowym
   * @param msg komunikat dodatkowy
   **/
  void gpib_error(std::string msg);

  /**
   * wysyla komunikat do uzadzenia. Jak się nie uda to wypisuje komunikat o błędzie
   * i kończy działanie programu.
   * @param cmd komunikat
   * @return true jak się uda
   **/
  bool wr(std::string cmd);

  /**
   * wysyła komunikat do urządzenia z bufora
   * @param buf bufor z którego chcemy czytać
   * @param nr liczba znaków do wysłania
   **/
  bool  wrBuf(void *buf,int nr=1024);

  /**
   * czyta odpowiedz od uzadzenia
   * @param nr maksymalna liczba znakow jaka chcemy odebrac
   **/
  std::string rd(int nr=1024);

  /**
```

```
   * czyta odpowiedz od uzadzenia, zapisuje do bufora
   * @param buf bufor do ktorego chcemy pisac
   * @param nr maksymalna liczba znakow jaka chcemy odebrac
   * @return ilos odczytanych znakow
   **/
  int  rdBuf(void *buf,int nr=1024);

  /**
   * wysyla komunikat do uzadzenia i czeja na odpowiedz
   * @param cmd komunikat
   * @param nr maksymalna liczba znakow jaka chcemy odebrac
   **/
  std::string query(std::string cmd,int nr=1024);

  /**
   * zrzuca informacjie z GPIB do pliku
   * @param filename nazwa pliku
   **/
  bool GPIBtoFile(std::string filename);

  /**
   * zrzuca plik na GPIB
   * @param filename nazwa pliku
   **/
  bool FiletoGPIB(std::string filename);

public:

  /**
   * Ustawia nazwe producenta (wazne przed autoconnect)
   * @param ManufactorName nazwa producenta
   **/
  void setManufactorName(std::string ManufactorName);

  /**
   * Ustawia nazwe uzadzenia (wazne przed autoconnect)
   * @param DeviceName nazwa uzadzenia
   **/
```

```
void setDeviceName(std::string DeviceName);

/**
 * Probuje sie polaczyc z uzadzeniem pod wczesniej ustawionym adresem (pa,sa,bi)
 **/
virtual bool connect(void);

/**
 * Probuje sie polaczyc z uzadzeniem o zadanym adresie
 **/
virtual bool connect(int pa,int sa=0,int bi=0);

/**
 * Sprawdza czy uzadzenie jest poprawnie podlaczone
 * tzn. czy m_ManufactorName uzadzenia m_DeviceName zgadzaja sie z tymi
 * ustawionymi w klasie.
 **/
virtual bool check(void);

/**
 * Probuje znalezc adres uzadzenia ktore sie przedstawia uzywajac
 **/
virtual bool autoConnect();

/**
 * Resetuje uzadzenie
 **/
virtual void clearDevice(void);
```

Dodatek C

Licencja

Licencja dla programu Ptolemy II. Pochodzi ona z Uniwersytetu w Berkeley. Jest dużo bardziej liberalna, niż na przykład GPL.

Permission is hereby granted, without written agreement and without license or royalty fees, to use, copy, modify, and distribute this software and its documentation for any purpose, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

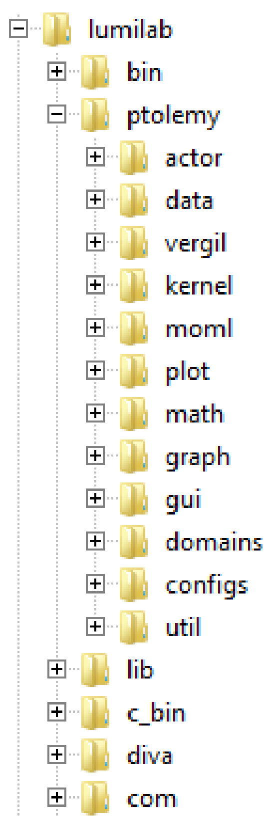
IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN ”AS IS” BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS

Dodatek D

Struktura katalogów

Struktura katalogów.



Rysunek D.1. *Struktura katalogów w środowisku GUI.*

W katalogu *bin/* znajdują się skompilowane pliki GUI(to zniego należy uruchamiać środowisko odpalając skrypt starotowy). W katalogu *ptolemy/* znajduje się kod źródłowy, nazwy podkatalogów są dobrane tak aby wiadome było co w danym podkatalogu się znajduje. Wspomnieć tylko należy, iż w podkatalogu *domains/* znajdują się pliki do obsługi rezyserów, natomiast w katalogu *configs/* pliki konfiguracyjne.

W katalogu *lib/* znajdują sie skompilowane biblioteki wykorzystywane przez środowisko. Katalog *cbin/* zawiera binarne wersje steorowników obsługujących urządzenia pomiarowe. Katalog *diva/* jak nie trudno się domyśleć zawiera środowisko Diva wykorzystywane do budowy interfejsu graficznego. W katalogu *com/* znajdują się narzedzia pomocnicze, typu JLex etc.

Bibliografia

[1] IEEE Std 488.2-1992 IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation -Description

[2] http://ostatic.com/libgpib

[3] http://tclap.sourceforge.net/

[4] HP4195A Maintance manual

[5] HP4195A Operation manual

[6] HP4395A Operation Manual

[7] B1500A User manual

[8] B1500A Programming manual

[9] HP4284A Operation manual

[10] HP4145B Operation manual

[11] HP6624A Programming manual

[12] SMB 100 Operation manual

[13] AWG2021 User manual

[14] AWG2021 Progrraming manual

[15] AFG3102 User manual

[16] TDS User manual

[17] http://www.gtk.org/

[18] http://www.qtsoftware.com/

[19] http://www.gnu.org/licenses/licenses.pl.html

[20] www.ruby-lang.org

[21] www-cad.eecs.berkeley.edu/diva/

[22] ptolemy.eecs.berkeley.edu/

[23] ptolemy.berkeley.edu/conferences/01/src/.../51neuendor.pd

[24] C. Hewitt, "Viewing control structures as patterns of passing messages," Journal of Artificial Intelligence, 8(3):323–363, June 1977.

[25] G. Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, MA, 1986.

[26] G. Agha, "Concurrent object-oriented programming," Communications of the ACM, 33(9):125–140, Sept. 1990.

[27] G. Agha, I. A. Mason, S. F.Smith, and C. L. Talcott, “A foundation for actor computation. Journal of Functional Programming, 7(1):1–72, 1997.

[28] G. Agha, C. Hewitt, „Concurrent programming using actors: Exploiting large-scale parallelism’, A.I Memo no 865, MIT Press, Cambridge, MA, 1985

[29] kepler-project.org/

[30] D. Przyborowski , „Development of General Purpose Low–power Small–area 10 bit CMOS DAC „, Proc. MIXDES2009

[31] http://nuclear.gla.ac.uk/twiki/pub/Main/Panda.../1-1.Idzik-Marek.Lumical-asic.pdf

[32] www.linearcollider.org/

[33] www.hamamatsu.com/

[34] M. Idzik, K. Świentek, S. Kulis Development of Pipeline ADC for the Luminosity Detector at ILC . Proceedings of the 15th International Conference ”Mixed Design of Integrated Circuits and Systems” MIXDES 2008, Poznan, Poland, 19-21 June 2008

[35] IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters, The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, NY 10016-5997, USA , 2001

[36] M. Idzik, K. Swientek, S. Kulis Progress report on LumiCal front-end electronics development. EUDET Annual Meeting, 8-10 October, Paris 2007

Spis rysunków

1.1. *Port GP-IB zaznaczony ramką na czerwono.* 13
1.2. *Z lewej: Schematyczny widok na przekrój przewodu. Z prawej: Przewód GP-IB* 14
1.3. *Sieć urządzeń może być tworzona w sposób liniowy(lewy obrazek), lub w gwiazdę (prawy obrazek).* 15
1.4. *Kontroler GP-IB stosowany w laboratorium. Do komputera PC podłączany za pomocą portu USB.* 15

2.1. *Architektura środowiska Diva. Istnieje struktura warstwowa, na samym dole zbudowany jest rdzeń na którym opiera się oprogramowanie. Dalej istnieje warstwa danych, które są obrabiane. Na samej górze znajduje się użytkownik, który poprzez interfejs steruje całością.* 35

3.1. *Asymetryczność działań aktorów* 37
3.2. *Przykład realizacji poprzez listę* 38
3.3. *Przykład realizacji poprzez wektor* 38
3.4. *a) programowanie sekwencyjne b) współbieżne(wielowątkowe).* 39

4.1. *Struktura budowanych systemów pomiarowych uwzględniająca środowisko graficzne.* . . . 42
4.2. *Przykładowa hierarchia stworzona poprzez pliki konfiguracyjne XML.* 44
4.3. *Okno powitalne, umieszczone w nim są podstawowe informacje na temat programu.* . . . 46
4.4. *Okno tworzenia modelu.* 47
4.5. *Okno zapisywania modelu do pliku.* 48
4.6. *Okno otwierania modelu z pliku.* 48
4.7. *Po stworzeniu pliku Java i umieszczeniu informacji w pliku XML, aktor pojawia się w odpowiednim menu.* 56
4.8. *W celu stworzenia aktora, należy zaznaczyć obszar z aktorami, które stanowią jego wnętrze. Następnie wybrać z menu „VIEW" opcję „COMPOSITE ACTOR".* 57
4.9. *Każdy stworzony aktor dostępny jest poprzez „USER LIBRARY".* 57
4.10. *Opcje dla reżysera.* 59
4.11. *Ogólna struktura środowiska GUI w oparciu o narzędzie Ptolemeusz.* 59
4.12. *Przykładowy widok na okno opcji dla wybranego aktora.* 60
4.13. *Łączenie portów wybranych aktorów.* 61

5.1. *Schemat stanowiska do pomiarów statycznych i pomiarów mocy układów DAC.* 64

5.2. *Cykle zegara generowanego poprzez AWG z zaznaczonym miejscem pomiaru analizatora B1500A.* . . . 64
5.3. *Model do pomiarów statycznych układów DAC.* . . . 65
5.4. *Parametry ustawiane przez użytkownika dla aktora „setMeasurement”.* . . . 66
5.5. *Parametry ustawiane przez użytkownika dla aktora „setTrigger”.* . . . 66
5.6. *Funkcja przenoszenia dla kanału numer 1 badanego układu.* . . . 67
5.7. *Wyniki zbiorcze dla wszystkich pięciu kanałów. Wykres przedstawia jedynie wycinek z całości zebranych danych.* . . . 68
5.8. *DNL i INL dla wybranych kanałów.* . . . 69
5.9. *Schemat stanowiska pomiarowego do pomiarów I-V sensorów.* . . . 70
5.10. *Model wykorzystywany w pomiarach I-V sensorów.* . . . 70
5.11. *Parametry do nastawiania przez użytkownika dla aktora „sweepDevMeasure”.* . . . 71
5.12. *Wyniki dla padu numer 63.* . . . 72
5.13. *Wyniki dla padu numer 64.* . . . 72
5.14. *Schemat stanowiska pomiarowego do pomiarów C-V sensorów.* . . . 73
5.15. *Model dla pomiarów C-V.* . . . 73
5.16. *Budowa aktora pochodnego wykorzystywanego w modelu do pomiarów C/V.* . . . 74
5.17. *Opcje dla aktora „hp4284a-measureConfiguration”.* . . . 75
5.18. *Opcje dla aktora „b1500-setVoltage”.* . . . 75
5.19. *Opcje dla aktora „b1500-sweepVoltage”.* . . . 76
5.20. *Wyniki pomiarów C-V dla pad numer 63.* . . . 76
5.21. *Wyniki pomiarów C-V dla pad numer 64.* . . . 77
5.22. *Schemat stanowiska pomiarowego do badania generatorów.* . . . 78
5.23. *Podział pasma całkowitego na N podpasm* . . . 79
5.24. *Całka z sygnału i szumów. Sposób obliczania.* . . . 79
5.25. *Model do pomiarów parametrów generatorów.* . . . 80
5.26. *Opcje dla aktora „Noise”.* . . . 81
5.27. *Widmo dla generatora AWG2021 prz amplitudzie 0.2V i f=100kHz.* . . . 82
5.28. *Widmo dla generatora AWG2021 przy amplitudzie 0.2V i f=1MHz.* . . . 82
5.29. *Widmo dla generatora AWG2021 przy amplitudzie 1V i f=1Mhz.* . . . 83
5.30. *Widmo dla generatora AWG2021 przy amplitudzie 2V i f=1MHz.* . . . 83
5.31. *Widmo dla generatora AFG3102 przy amplitudzie 0.2V i f=1MHz.* . . . 84
5.32. *Widmo dla generatora AFG3102 przy amplitudzie 1V i f=1MHz.* . . . 85
5.33. *Widmo dla generatora AFG3102 przy amplitudzie 2V i f=1Mhz.* . . . 85
5.34. *Widmo dla generatora AFG3130 przy amplitudzie 0.2V i f=10MHz.* . . . 86
5.35. *Widmo dla generatora AFG3130 przy amplitudzie 1V i f=10MHz.* . . . 86
5.36. *Widmo dla generatora AFG3130 przy amplitudzie 2V i f=10MHz.* . . . 87
5.37. *Widmo dla generatora SMB100 przy amplitudzie 0.2V i f=1MHz.* . . . 88
5.38. *Widmo dla generatora SMB100 przy amplitudzie 1V i f=1MHz.* . . . 88
5.39. *Widmo dla generatora SMB100 przy amplitudzie 2V i f=1MHz.* . . . 89
5.40. *Widmo zbiorczegeneratora SMB100 i f=1MHz.* . . . 89

5.41. *Widmo dla generatora SMB100 przy amplitudzie 0.2V i f=10MHz.* 90
5.42. *Widmo dla generatora SMB100 przy amplitudzie 1V i f=10MHz.* 90
5.43. *Widmo dla generatora SMB100 przy amplitudzie 2V i f=1MHz.* 91

D.1. *Struktura katalogów w środowisku GUI.* 103

Spis tablic

1.1. *Tabela przedstawiająca linie sterujące w standardzie GP-IB. Pierwsza kolumna - nazwa rozkazu, druga - tryb pracy urządzenia do jakiego jest wysyłany, trzecia - krótki opis.* . . 14
1.2. *Zbiór standardowych komend według standardu IEEE 488.2, które wspierać muszą wszystkie urządzenia z nim zgodne.* 16
1.3. *Metody z klasy GpibDev ułatwiające tworzenie oprogramowania dla urządzeń GP-IB.* . . 17
1.4. *Paremtry i argumenty dla analizatora HP4195A.* 23
1.5. *Paremtry i argumenty dla analizatora HP4395A.* 24
1.6. *Paremtry i argumenty dla analizatora urządzeń półprzewodnikowych B1500A.* 26
1.7. *Paremetry i argumenty dla miernika LCR HP4284A.* 27
1.8. *Paremetry i argumenty dla analizatora urządzeń półprzewodnikowych HP4145B.* 28
1.9. *Paremetry i argumenty dla zasilacza HP6624A.* 28
1.10. *Paremetry i argumenty dla generatora SMB100.* 29
1.11. *Paremetry i argumenty dla generatora AWG2021.* 29
1.12. *Paremetry i argumenty dla generatora AFG3102.* 30
1.13. *Paremetry i argumenty dla oscyloskopów TDS firmy TEKTRONIX.* 31

4.1. *Najważniejsze pliki konfiguracyjne.* 42
4.2. *Skróty klawiszowe dostępne w oknie „GRAPH”* 48

5.1. *Przykładowe wyniki pomiarów mocy za pomocą stworzonego oprogramowania.* 67
5.2. *Obliczenia wartości SNHR i SINAD dla generatora AWG2021.* 84
5.3. *Obliczenia wartości SNHR i SINAD dla generatora AFG3031.* 87
5.4. *Obliczenia wartości SNHR i SINAD dla generatora SMB100.* 91