

Uniwersytet Warszawski

Wydział Fizyki

**Systemowe podejście do testowania elektroniki
wyzwalania w eksperymentach wysokiej energii**

Michał Pietrusiński



**Praca magisterska
napisana pod kierunkiem**

Prof. dr hab. Jana Królikowskiego

Warszawa 1998

Spis treści

I. WSTĘP.....	1
II. CHARAKTERYSTYKA ŚRODOWISKA ELEKTRONICZNEGO.....	4
A. FAZA PROJEKTOWANIA	4
B. FAZA DOCELOWA	4
III. RODZAJE TESTÓW	6
IV. STANDARDOWE NARZĘDZIA TESTOWE	10
A. FDPM	10
B. JTAG	11
1. Ogólnie	11
2. Podstawa działania	13
3. Po co tego używać	16
V. PODSUMOWANIE.....	19
DODATEK A. TESTY W BARI	20
1. Wstęp	20
2. Algorytm działania trygera RPC	20
3. Stanowisko testowe	21
4. Przebieg i wyniki testów.....	25
5. Oprogramowanie	30
6. Konkluzje	35
DODATEK B. PROGRAMY DO OBSŁUGI FDPM.....	37
1. Wstęp	37
2. Składnia skryptów fsc.....	37
3. Przegląd programów	42
4. Implementacja	44
DODATEK C. JTAG SOFTWARE	46
1. Wprowadzenie	46
2. Model warstwowy	48
3. Obecny stan prac	51

4. Podsumowanie.....	59
DODATEK D. TESTY PROCESORA PAC	61
1. Wstęp	61
2. Architektura układu	61
3. Stanowisko testowe	65
4. Oprogramowanie do testowania	66
5. Strategia testów	68
6. Wyniki testów i podsumowanie	73
PODZIĘKOWANIA	74
BIBLIOGRAFIA	75
INDEX	77

I. Wstęp

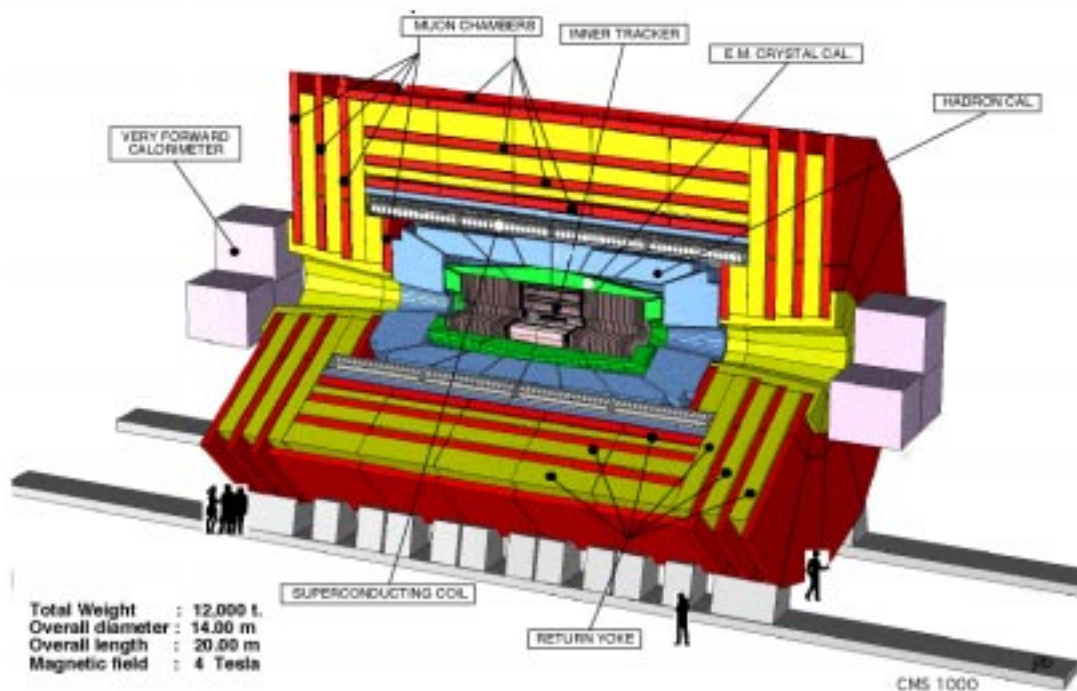
Systemy wyzwania (tzw. trygery) w eksperymentach wysokiej energii spełniają niezwykle istotną funkcję: spośród dziesiątek milionów zderzeń na sekundę wybierają te, które są ważne z fizycznego punktu widzenia dając sygnał do systemu akwizycji danych, by nastąpiło zachowanie przypadku do późniejszej analizy. Błędy w ich działaniu wystawiają na ryzyko powodzenie eksperymentu, stwarzając niebezpieczeństwo zebrania nieadekwatnych danych lub w ogóle uniemożliwienia zbierania danych. Z tego powodu należy położyć szczególny nacisk na dokładne testowanie tych systemów oraz kontrolę ich sprawnego funkcjonowania, czemu poświęcona jest ta praca. Spostrzeżenia tu zawarte są owocem kilkuletniej pracy autora przy testowaniu prototypów projektowanego trygera RPC dla detektora CMS przy akceleratorze LHC w CERN, jak również przy pisaniu oprogramowania dla komponentu VETO WALL detektora ZEUS przy akceleratorze HERA. Autor jest współtwórcą oprogramowania opisanego w dodatkach do tej pracy.

Detektor CMS [29] pojawia się w pracy jako przykład detektora będącego w fazie projektowania. Wzmianki dotyczące detektora ZEUS obrazują pewne stwierdzenia na podstawie sprawnie działającego już detektora.

Detektor CMS (Rysunek 1) został zaprojektowany w taki sposób, aby zoptymalizować detekcję mionów. W tym celu, po rozważeniu wielu konfiguracji pola magnetycznego, ze względu na wymóg zwartej budowy, zdecydowano się na użycie solenoidu - pole magnetyczne przezeń wytwarzane jest najsilniejsze. Długi, 13 metrowy nadprzewodzący solenoid o promieniu wynoszącym 3 metry, wytwarzający pole o indukcji do 4 T, powinien zagwarantować dobrą rozdzielczość pędową dla mionów o wysokich pędach (≈ 1 TeV).

Mionowa część detektora CMS obejmuje obszar do $|\eta|=2.4$ odpowiadający kątom biegunowym z zakresu od 9.4 do 170.6 stopni. Tak duży obszar akceptacji potrzebny jest m.in. ze względu na przewidywane rozpady lekkich cząstek Higgsa na cztery miony ($H \rightarrow ZZ^* \rightarrow \mu^+ \mu^- \mu^+ \mu^-$). W celu zapewnienia precyzyjnego pomiaru pędu mionów od kilku GeV do kilku TeV detektor składa się z systemu czterech koncentrycznych powierzchni zawierających stacje mionowe poprzedzielanych płytami z żelaza, które spełniają zarówno rolę absorbenta jak i zamykają strumień pola magnetycznego. Stacje mionowe umieszczone są w beczce i endcapie detektora. Każda ze stacji w beczce składa się z modułu komór dryfowych (DT): dają one precyzyjny pomiar pozycji przejścia mionu ($\approx 100 \mu\text{m}$) przy spodziewanej tam małej częstości ($< 10 \text{ Hz/cm}^2$). W endcapie ze względu na większe częstości oraz niestałe pole magnetyczne zostaną zastosowane komory

CSC (ang. Cathode Strip Chamber). Ponadto wszystkie stacje posiadają przynajmniej po jednej powierzchni komór RPC (ang. Resistive Plate Chamber, [29,30]), które co prawda mają mniejszą rozdzielczość przestrzenną, ale dużo lepszą rozdzielczość czasową niż komory DT i CSC. Pozwolą one na jednoznaczna identyfikację przecięcia, z którego pochodzą zarejestrowane miony.



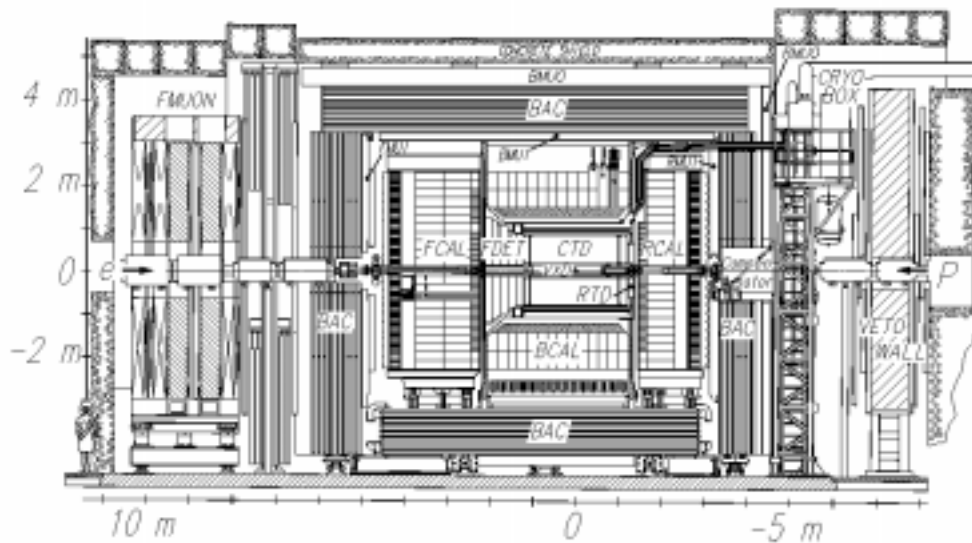
Rysunek 1. Detektor CMS

Tryger mionowy powinien w krótkim czasie (około trzech mikrosekund) umieć identyfikować miony, mierzyć ich pęd poprzeczny, oraz przypisać przecięcie, z którego one pochodzą. Łączy on informacje pochodzące od trygera komór DT (CSC) oraz trygera RPC i przekazuje do trygera pierwszego stopnia. Tryger 1. stopnia musi zredukować liczbę zdarzeń z 10^9 na sekundę do częstości 100 kHz (czynnik 10^4). Odpowiedź trygera 1. stopnia musi następować najpóźniej 3 μ s od zajścia zdarzenia.

W budowę trygera 1. stopnia zaangażowanych jest wiele grup z państw będących członkami kolaboracji CMS. W szczególności grupa CMS z Uniwersytetu Warszawskiego oraz Instytutu Problemów Jądrowych im. Andrzeja Sołtana odpowiedzialna jest za projekt trygera mionowego RPC.

Celem detektora ZEUS (Rysunek 2) jest precyzyjny pomiar energii, kierunków oraz natury pojedynczych cząstek oraz jetów powstałych w oddziaływaniach elektron-proton. Komponent VETOWALL stanowi element trygera pierwszego stopnia. Zbudowany jest z dwóch powierzch-

ni liczników scyntylacyjnych umiejscowionych przy wylocie wiązki protonowej, których zadaniem jest wykrycie cząstek pochodzących z halo wiązki protonowej.



Rysunek 2. Detektor ZEUS

Jeżeli w odpowiednim czasie przed zderzeniem detektor VETOWALL rejestruje cząstki, które nie mogą pochodzić z poprzedniego zderzenia, przypadek taki jest odrzucany.

II. Charakterystyka środowiska elektronicznego

Podstawową cechą środowiska elektronicznego systemów wyzwania jest ich ogromne zróżnicowanie. Testowanie tych systemów powinno obejmować dwie fazy: fazę projektowania tych systemów oraz fazę docelową.

A. Faza projektowania

W fazie projektowania nie posiadamy jeszcze pełnego, skonkretyzowanego, stabilnego systemu. Budowane są natomiast prototypy pewnych jego elementów, bardzo często nie działające poprawnie i często modyfikowane. Składają się one zazwyczaj z maksymalnie kilku płyt elektronicznych. Nie da się zawnazu dokładnie przewidzieć, jak wyglądać będą przyszłe elementy – zależy to od rozwoju projektu.

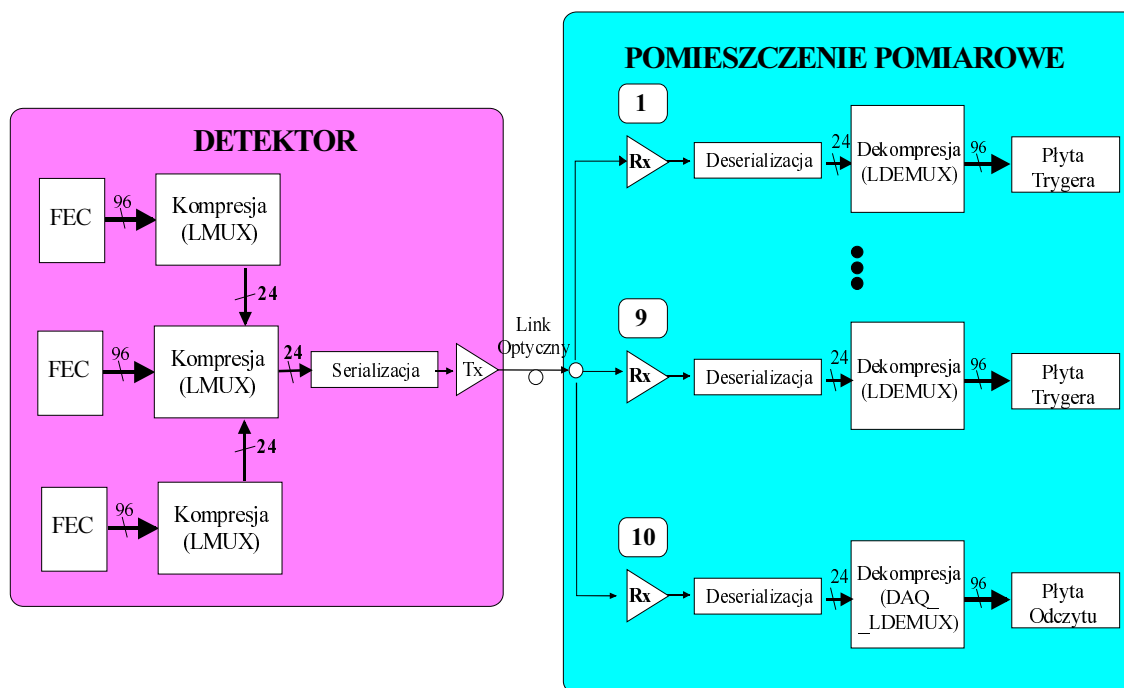
Przykładem niech będzie stanowisko testowe do badania trygera mionowego RPC dla detektora CMS [1,2,3]. Składa się ono z kasety VME połączonej za pomocą interfejsu Bit3 ze stacją roboczą Sun i urządzeń towarzyszących, jak oscyloskop cyfrowy, analizator stanów logicznych, generator impulsów itp. Na tym stanowisku testowanych było kilka prototypów elementów przyszłego systemu trygera jak np. synchronizator, procesor PAC (Pattern Comparator) zrealizowany w układach Altera, sorter, system odczytu. Elementy te stanowiło parę płyt VME: płyty wejściowe, płyta synchronizatora, płyta trygera, płyty odczytu oraz płyta kontrolna. Podczas pracy prototypów okazało się, że w celu dokładniejszej analizy niektóre z elementów musiały zostać zmodyfikowane. Dodano dodatkowo możliwość odczytu kodów generowanych przez procesory PAC przed wejściem na układ sortujący, oraz możliwość użycia synchronizatora jako układu symulującego prawdziwe impulsy z komór. Oczywiście zmiany te pociągnęły za sobą szereg modyfikacji istniejącego oprogramowania. Prace te opisane są szerzej w Dodatku A.

B. Faza docelowa

Elektronika w fazie docelowej ma zupełnie inny charakter niż w fazie projektowania. Nie podlega już ona tak częstym modyfikacjom. Stanowi ona olbrzymi, szybki, wielokanałowy, potokowy system zbudowany z wielu elementów, przy czym część z nich jest do siebie bardzo podobnych. Część urządzeń jest rozmieszczona na detektorze, gdzie nie ma do nich fizycznego dostępu.

System trygera mionowego RPC [3,4,5] składać się będzie z szeregu układów umieszczonych bezpośrednio na komorach RPC realizujących czytanie analogowych sygnałów z komór i konwersję na sygnały cyfrowe (układy FEC, ang. Front End Chip), synchronizację, kompresję

(LMUX), serializację i nadanie przez łączy optyczne, oraz elektroniki znajdującej się w pomieszczeniu pomiarowym (kilkadziesiąt kaset VME), której zadaniem jest odebranie sygnałów, dekompresja (LDEMUX) oraz wykonanie algorytmu trygera (Rysunek 3).



Ignacy Maciek Kudła, Warsaw University

Rysunek 3. Tryger mionowy RPC - schemat blokowy.

III. Rodzaje testów

Systemy wyzwania w eksperymentach wysokiej energii są urządzeniami bardzo skomplikowanymi i złożonymi. W systemach tych jest rzeczą absolutnie pewną, że będą występowały pewne nieprawidłowości związane z błędami w projekcie, złym ustawieniem parametrów pracy, przesłuchami, wadami mechanicznymi, które mogą powstać w każdym momencie, itp. Systemy te pełnią bardzo ważną funkcję w eksperymencie. Od jakości ich funkcjonowania zależy trafność zebranych danych. Nie do dopuszczenia jest sytuacja, żeby po kilku miesiącach trwania eksperymentu okazało się, że zebrane dane nie odpowiadają warunkom narzuconym na tryger. Źle by było również, gdyby w momencie odkrycia nieprawidłowości w działaniu trygera należało budować specjalne narzędzia pozwalające zlokalizować przyczynę błędów. Z tych względów elektronika trygera powinna być szczegółowo testowana już w fazie projektowania. Ponadto w system powinny być wbudowane narzędzia kontrolujące poprawność zbieranych danych i pozwalające na precyzyjne określenie miejsca powstawania potencjalnych błędów.

Od strony fazy, w jakiej znajduje się elektronika (projekt) oraz czasu i wnikliwości testów, testy trygera można z grubsza sklasyfikować na:

(a) testy prototypów docelowych elementów (faza projektowania)

Zadaniem tych testów jest weryfikacja algorytmu działania projektowanego elementu, porównanie rezultatów symulacji z działaniem elektroniki, poszukiwanie problemów, których na etapie symulacji nie dało się przewidzieć. Przykładem takich testów są prace opisane w dodatkach A i D oraz w [1,3,4].

(b) testy docelowych elementów w laboratorium

Przed montażem na detektorze, elementy muszą zostać gruntownie przetestowane w laboratorium. Celem tych testów jest znalezienie wszystkich uszkodzeń mechanicznych, wad produkcyjnych, zwarć, przesłuchów itd. Podobne testy powinny być również przeprowadzane w czasie dłuższych przerw w działaniu detektora. Wskazany wydaje się przygotowanie bibliotek zawierających standardowe wektory testowe dla pojedynczych elementów lub ich grup. Część takich wektorów uzyskać można od producentów elementów, np. pamięci. Pozostałe wektory testujące dedykowaną elektronikę mogą zostać wygenerowane przy użyciu specjalnego oprogramowania, np. analizatorów połączeń na płytkach drukowanych (por. Dodatek C), symulatorów układów programowalnych (np. Altera® MAX-PLUS II), symulatorów VHDL itd. W celu aplikacji wektorów testowych posłużyć się można mechanizmami Boundary-Scan (IV.B, Dodatek C). Do przeprowadzenia dyna-

micznych testów funkcjonalnych niezbędne jest zbudowanie odpowiednich stanowisk testowych pozwalających na aplikację pobudzeń (np. przy użyciu FDPM – patrz IV.A) i odczyt reakcji urządzenia.

Przykładem takiego stanowiska jest stanowisko testowe do badania elementów detektora VETO WALL. Pozwala ono na wykonanie predefiniowanych testów płyt realizujących najważniejsze funkcje detektora. Można również za jego pomocą wykonywać operacje nie-standardowe i z łatwością modyfikować predefiniowane testy. Składa się ono z kasety VME, transputera, komputera PC oraz zasilaczy, oscyloskopu, mierników itd. Oprogramowanie transputera, które można łatwo zmodyfikować przy pomocy komputera PC, wykonuje predefiniowane testy płyt. W główne elementy trygera są natomiast wbudowane narzędzia pozwalające symulować część rzeczywistych sygnałów pojawiających się podczas trwania eksperymentu.

(c) krótkie testy systemu przed użyciem detektora

Przed każdym użyciem detektora system powinien być poddany krótkim testom sprawdzającym, czy zostały włączone zasilania, prawidłowo połączone kable, itd. Mogą też być przeprowadzone krótkie testy własne (ang. self-test) wbudowane w urządzenia elektroniczne (np. IEEE 1149.1 Built-In Self-Test, patrz IV.B, Dodatek C), oraz szybkie testy połączeń na płytkach (np. IEEE 1149.1 Virtual Interconnect Test, patrz Dodatek C).

W detektorze VETO WALL przed każdym runem testowany jest główny element trygera – tzw. macierz koincydencji. Ostatnio dodano również wykonywanie zaawansowanych testów własnych dla systemu Hit-Readout, kontrolującego jakość działania trygera.

(d) kontrola i testy systemu podczas trwania eksperymentu

Zależą one od funkcji spełnianych przez system i możliwości testowych wejść wbudowanych. Można je podzielić na [12]:

- kontrolę quasi-bieżącą stanu aparatury, czyli monitorowanie najważniejszych parametrów pracy aparatury elektronicznej. Dla VETO WALL są to m.in. wartości napięć przyłożonych do fotopowielaczy oraz wartości progów dyskryminatorów.
- kontrolę quasi-nadążną danych, czyli monitorowanie pewnej ilości danych (np. co n -ty przypadek), które są analizowane podczas naboru. W detektorze VETO WALL kontroluje się w ten sposób np. częstość uderzeń cząstek w liczniki.
- nadążną kontrolę danych, która polega na kontroli dla każdego przypadku wybranych danych. W detektorze ZEUS kontroluje się na przykład liczniki przypadków dla GFLT

(Global First Level Trigger), GSLT (Global Second Level Trigger) oraz BCN (Bunch Crossing Number) i porównuje się ich zawartości z odpowiednimi wartościami w paczkach danych przychodzących od poszczególnych komponentów, co umożliwia kontrolę i synchronizację danych pochodzących od różnych komponentów. W systemie Hit-Readout kontroluje się dla każdego przypadku wiele parametrów, m.in. formaty zapisywanych danych oraz szereg liczników.

Należy zaznaczyć, że stopień w jakim można testować elektronikę, zależy w największej mierze od narzędzi testowych wbudowanych w sam system. Dotyczy to zarówno spojrzenia globalnego (testowanie systemu jako całości), jaki i pewnych jego elementów, aż do konkretnych układów scalonych włącznie [11]. Systematyczne metody budowania testowalnej i kontrolowalnej elektroniki dla eksperymentów wysokiej energii w oparciu o Model Elementarnego Bloku Funkcjonalnego szerzej opisane są w pracy [12]. System zbudowany w oparciu o ten model charakteryzuje się zespoleniem jego części funkcjonalnej z częścią kontrolno-diagnostyczną, możliwością wykorzystania potoku danych fizycznych jako źródła informacji o bieżącym stanie systemu oraz bogatymi możliwościami diagnostyczno-testowymi.

Również oprogramowanie powinno być pisane z myślą o istniejących w nim błędach oraz być przygotowane na istnienie nieprawidłowości w części elektronicznej systemu. W przypadku wykrycia błędu programista powinien otrzymywać wyczerpujące informacje na temat okoliczności jego wystąpienia, np. listę ostatnio wywołanych funkcji wraz z ich parametrami, stan obiektu, w którym wykryto błąd itd. Jeśli to możliwe, powinien składać się on z wielu niezależnych modułów, które mogą być testowane oddzielnie. Dobrą praktyką jest przygotowanie dla każdego modułu procedur testujących go, które warto wykonywać po każdej modyfikacji modułu.

Jeśli to możliwe, należy dążyć do tego, by jak najwięcej elementów związanych z testami (a), (b), (c) i (d) było wspólnych. Dotyczy to zarówno elektroniki jak i oprogramowania. Prowadzi to do zmniejszenia wkładu pracy w przygotowanie tych elementów i dokładne poznanie ich działania, pozwala uniknąć powstania wielu błędów i co najważniejsze prowadzi do sytuacji, w której warunki w jakich przeprowadza się testy są najbardziej zbliżone do warunków rzeczywistej pracy. Idealem byłoby, kiedy wszystkie testy można by przeprowadzić „na detektorze”. W przypadku VETO WALL istnieje szereg testów, które przeprowadza się w normalnych warunkach pracy trygera, jak np. omówione powyżej testy własne systemu Hit-Readout. Jednakże stanowisko testowe do przeprowadzania testów w laboratorium nie daje możliwości odtworzenia rzeczywistych warunków pracy i korzysta z niezależnego oprogramowania.

Celowym wydaje się poszukiwanie pewnych standardowych narzędzi umożliwiających testowanie elektroniki w jej różnych fazach. Problemowi temu został poświęcony następny paragraf.

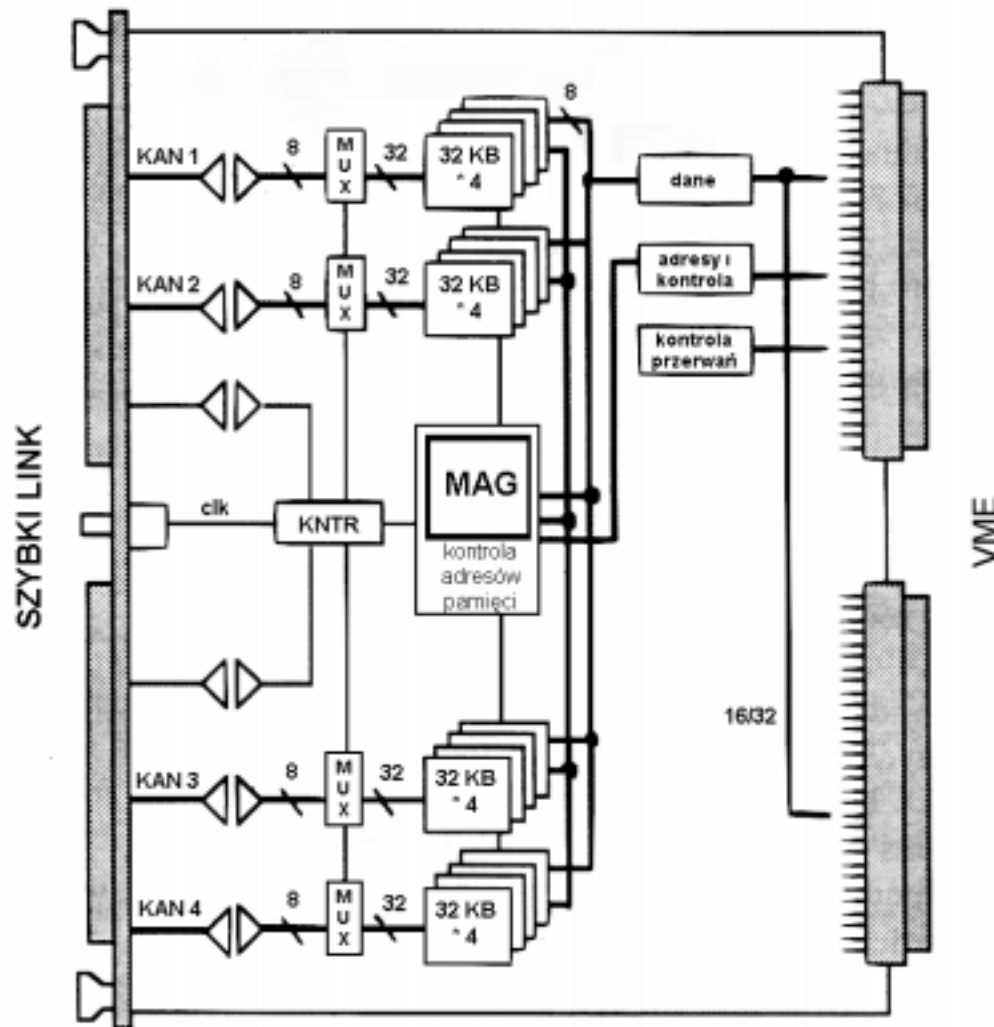
IV. Standardowe narzędzia testowe

W paragrafie tym opisano mniej znane narzędzia stosowane przez warszawską grupę CMS od kilku lat, nadające się do zastosowania przy testowaniu większych systemów elektronicznych, a w szczególności szybkich systemów przetwarzania danych w eksperymentach wysokiej energii.

A. FDPM

Systemy wyzwania w eksperymentach wysokiej energii pracują zazwyczaj w trybie przetwarzania synchronicznego, potokowego grup sygnałów o wysokich częstotliwościach. W celu skutecznego testowania elementów tych systemów konieczna jest umiejętność generowania i rejestracji takich sygnałów. O ile narzędzia do rejestracji są w przemyśle powszechnie dostępne (np. analizator stanów logicznych), to gorzej sprawa wygląda z ich generowaniem.

Narzędziem z powodzeniem stosowanym przez naszą grupę od kilku lat jest wyprodukowany przez CERN w ramach projektu RD12 FDPM (Fast Dual Port Memory) [7,8]. FDPM jest płytą VME, która może transmitować lub przyjmować dane z częstotliwością do 100 MHz przez cztery ośmiobitowe kanały, co daje przepustowość do 400 MB/sek (patrz Rysunek 4). Zawiera ona pół megabajta pamięci, układ scalony VLSI MAG (Memory Address Generator) [9], oraz szereg rejestrów. Szybkie wejścia/wyjścia połączone są równolegle z czterema blokami pamięci. Układ MAG w inteligentny sposób kontroluje adresowanie bloków pamięci. Bloki pamięci mogą być dostępne przez VME lub używane podczas transmisji sygnałów. Fizycznie każdy blok jest zaimplementowany w postaci czterech multipleksowanych elementów pamięci. W ten sposób prędkość przesyłania danych może być cztery razy szybsza niż prędkość dostępu do pamięci. Cztery bajty mogą być równolegle transmitowane lub przyjmowane z częstotliwością określoną przez zewnętrzny zegar (do 100 MHz). Rejestry są częścią programowalnej logiki kontrolnej. Określa się w nich m.in. adresy i długości rejonów pamięci, z których się chce korzystać. Rejony te mogą być wysyłane lub zapisywane po kolei lub w pętli. Zwiększa to znacząco długość możliwych do wygenerowania sekwencji sygnałów.



Rysunek 4. MAG – FDPM

Do obsługi FDPM wykorzystać można bibliotekę LabVIEW LabFDPM [10]. W celu wykorzystania FDPM do generowania sekwencji sygnałów stworzono programy *nfdp* i *xfdp* (Dodatek B). Napisano również bibliotekę w C++ (*libfdp*) umożliwiającą łatwe dodanie obsługi FDPM do innych programów.

Moduł FDPM był wielokrotnie stosowany przez naszą grupę do testowania prototypów elementów trygera mionowego (patrz Dodatek A, Dodatek B, Dodatek D, [4]).

B. JTAG

1. Ogólnie

Wzrastające wymagania co do testowalności nowoczesnej elektroniki są również stawiane

przez przemysł. Testowanie obejmuje dwa główne aspekty: kontrolę i obserwację. Aby przetestować system należy go wprowadzić w odpowiedni stan, dostarczyć dane testowe i obserwować w celu sprawdzenia, czy zachowuje się zgodnie z zamierzeniami. Tradycyjnie, dodawanie elektroniki zapewniającej odczyt, wytwarzanie wektorów testowych i ich wprowadzenie, oraz testy funkcjonalne znane są jako testowanie *ad hoc*. Zazwyczaj taki hardware testowy zwiększa znacząco takie parametry projektu, jak liczba elementów, pinów, powierzchnię, koszt, zużycie energii oraz wadliwość [13].

Postępy w metodologii testowania doprowadziły do rozwoju systemu zaproponowanego przez grupę JTAG (Joint Test Action Group). JTAG powstała w 1985 roku i liczyła ponad 200 członków z całego świata, obejmując głównych producentów elektroniki. System przez nią zaproponowany – Test Access Port and Boundary Scan Architecture – stał się wkrótce standardem przemysłowym (IEEE Std 1149.1-1990 [15]). Oferuje on jednolity mechanizm testowy dla różniących się od siebie działów przemysłu elektronicznego bez konieczności zrozumienia problemów związanych z innymi działami. W ten sposób projektant układu scalonego nie musi znać szczegółów dotyczących testowania np. płyt drukowanych. Musi on tylko wbudować w układ zgodne ze standardem narzędzie, które zapewni możliwość testowania układu, a jednocześnie płytki [16]. Standard pozwala na szeregowe wprowadzanie instrukcji i danych testowych do urządzenia oraz odczytywanie wyników za pomocą jedynie czterech dodatkowych pinów.

Standard może zostać wykorzystany na kilku poziomach w różnych etapach rozwoju projektu:

- na poziomie układów scalonych: standard ułatwia testowanie układów scalonych (aplikacja wektorów testowych, wykonywanie wbudowanych testów własnych, obserwacja sygnałów podczas pracy układu) [17], jak również umożliwia wprowadzanie danych do logiki układu lub ich czytanie (Dodatek C, Dodatek D);
- na poziomie płytek drukowanych: standard ułatwia testowanie płyt [20]; może być użyty do testowania płyt prototypowych [18], w trakcie produkcji [19], lub przed użyciem urządzenia;
- na poziomie modułu lub systemu: standard może być pomocny w celu testowania modułów zbudowanych z większej ilości elementów [21,22].

Standard pozwala na pracę w dwóch głównych trybach:

- nieinwazyjnym: udostępnione są wtedy elementy układu, które mogą działać niezależnie od jego jądra, nie zakłócając normalnej pracy układu. Można przy ich pomocy w asynchroniczny sposób odczytywać udostępnione sygnały jądra układu, przygotowywać testy i odczyty-

wać ich wyniki;

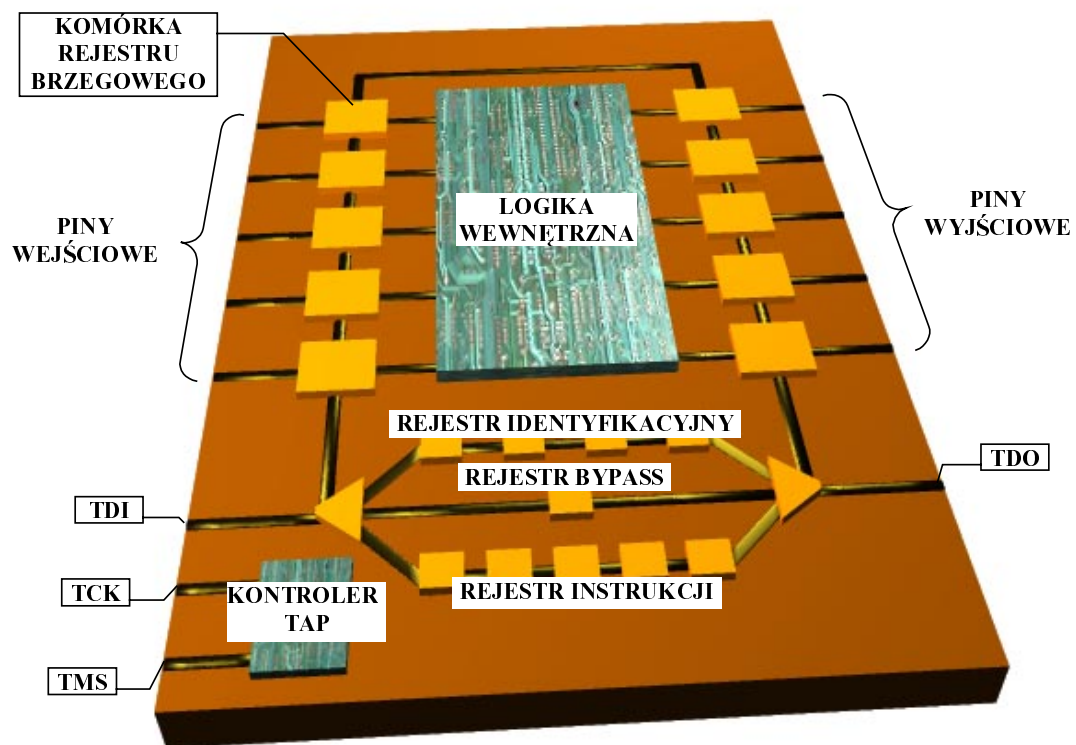
- z prawem dostępu do pinów: system testowy przejmuje kontrolę nad pinami układu odłączając je od jego wewnętrznej logiki. Pozwala to z jednej strony swobodnie testować wnętrze układu, z drugiej natomiast, obserwując lub wprowadzając piny w odpowiedni stan, testować jego otoczenie (piny, połączenia na płycie, sąsiednie układy [23] lub całe moduły).

Wielką zaletą standardu jest możliwość wprowadzania przez projektantów dodatkowych trybów pracy, zarówno nieinwazyjnych jak i z prawem dostępu do pinów, które pozwalają na wykorzystanie go do specyficznych dla danego układu testów.

2. Podstawa działania

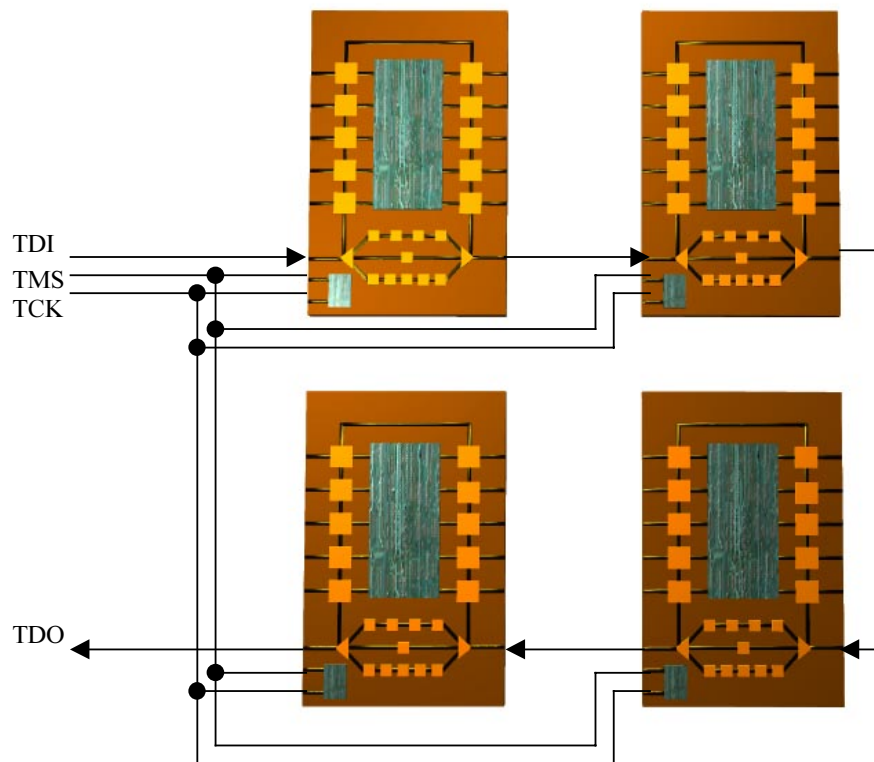
Podstawowe elementy obsługujące Boundary-Scan wbudowane są w układ scalony (patrz Rysunek 5). Są to:

- Cztery lub pięć dodatkowych pinów: TDI (Test Data In), TDO (Test Data Out), TMS (Test Mode Select) i TCK (Test Clock) oraz opcjonalny pin TRST* (Test Reset) tworzące standardową szynę TAP (Test Access Port);
- Kontroler TAP będący prostą maszyną stanów sterowaną sygnałami TMS oraz TCK, który generuje sygnały kontrolujące pozostałe elementy Boundary-Scan;
- Rejestr Instrukcji, który jest używany do ustawienia żadanego trybu pracy i wybrania odpowiedniego rejestru danych. Może on zostać podłączony przez kontroler TAP pomiędzy piny TDI oraz TDO w celu szeregowego wprowadzenia instrukcji lub odczytu danych kontrolnych (tzw. Capture Value). Część instrukcji jest narzucona przez standard (SAMPLE, EXTEST i BYPASS), część jest proponowana, ale opcjonalna (np. INTEST, IDCODE, HIGHZ, RUNBIST...). Istnieje również możliwość dodania własnych instrukcji.
- Grupa rejestrów określanych jako rejestry danych. W każdym układzie muszą być przynajmniej 2 rejestry danych: Rejestr Brzegowy (Boundary Register) oraz Rejestr Bypass (Bypass Register). Standard proponuje również umieszczenie opcjonalnych rejestrów, jak np. Rejestru Identyfikacyjnego i pozwala na dodanie własnych rejestrów. Rejestry danych umieszczane są pomiędzy TDI oraz TDO w zależności od instrukcji dekodowanej przez Rejestr Instrukcji oraz stanu kontrolera TAP.



Rysunek 5. Ogólny schemat układu zgodnego ze standardem IEEE 1149.1

Układy scalone zgodne ze standardem IEEE 1149.1 można ze sobą łączyć w taki sposób, by utworzyć jeden lub kilka łańcuchów testowych. Najprostsza konfiguracja polega na utworzeniu jednego, długiego łańcucha (Rysunek 6). Polega ona na połączeniu równoległym wszystkich pinów TMS i TCK (stan kontrolera TAP we wszystkich układach jest wtedy jednokowy), oraz pinów TDI i TDO w sposób szeregowy tak, by TDO poprzedniego układu w łańcuchu był połączony z TDI następnego układu. Rejestry Instrukcji i Rejestry Danych kolejnych układów tworzą wtedy w odpowiednich stanach Kontrolera TAP jeden rejestr przesuwany.



Rysunek 6. Połączenie układów w jeden łańcuch testowy

Typowy przebieg testu wygląda w następujący sposób. Najpierw Kontrolery TAP ustawiane są za pomocą odpowiedniej sekwencji sygnałów TCK i TMS do stanu Shift-IR. Rejestry Instrukcji wszystkich układów podłączone są wtedy pomiędzy TDI i TDO każdego z nich tworząc jeden długi rejestr przesuwany. Pozwala to na szeregowe wprowadzenie do nich za pomocą TDI oraz TCK żądanych instrukcji, definiujących rodzaj testu jaki ma zostać przeprowadzony. Niech będzie to dla przykładu instrukcja SAMPLE dla pierwszego układu, pozwalająca odczytać stany wszystkich pinów podczas jego normalnej pracy, nie zakłócając jego działania, oraz instrukcja BYPASS dla pozostałych układów wybierająca jako rejestr danych jednobitowy Rejestr Bypass. Następnie Kontrolery TAP wprowadzane są w stan Capture-DR. Stany pinów pierwszego układu są wtedy zapamiętywane w komórkach rejestru brzegowego. Po ustawieniu Kontrolerów TAP w stan Shift-DR, odpowiednie rejestry danych, w tym przypadku Rejestr Brzegowy dla pierwszego układu i Rejestry Bypass dla pozostałych, podłączane są do pinów TDI oraz TDO, co umożliwia odczytanie za pomocą pinów TDO i TCK zapamiętanych w pierwszym układzie danych.

Każdy układ zgodny ze standardem IEEE 1149.1 musi obsługiwać instrukcje SAMPLE/PRELOAD, BYPASS i EXTEST. Instrukcje SAMPLE/PRELOAD i BYPASS zosta-

ły już pokrótce omówione. Instrukcja EXTEST jest kluczową instrukcją pozwalającą na przeprowadzanie testów połączeń pomiędzy układami i modułami złożonymi z większej ilości układów. Podczas jej wykonywania logika wewnętrzna układu jest odłączona od pinów. W stanie Capture-DR Kontrolera TAP piny wyjściowe układu ustawiane są zgodnie z zawartością rejestru brzegowego, który można wstępnie załadować korzystając z instrukcji SAMPLE/PRELOAD, a stan pinów wejściowych jest w nim zapisywany. Zapewnia to pełną kontrolowalność i obserwowalność pinów układu: każdy pin wyjściowy może być ustawiony w dowolny stan, a stan każdego pinu wejściowego odczytany. Typowy test połączeń polega na ustawianiu różnych wartości na pinach sterujących układów i odczytywaniu stanów pinów wejściowych. W ten sposób znaleźć można przerwania ścieżek, niedokładne luty, zwarcia, jak również testować połączenia pomiędzy płytkami [13,14,16,19,22]. Możliwe jest także testowanie układów bez BS, jeżeli są one otoczone układami zgodnymi ze standardem IEEE 1149.1 [20,23].

Standard proponuje dodanie i opisuje parę opcjonalnych instrukcji, jak np.

- IDCODE, która wybiera rejestr identyfikacyjny zawierający dane identyfikujące rodzaj układu. Pozwala to sprawdzić, czy podczas montażu płytki typy układów nie zostały pomyłone,
- INTEST, która działa analogicznie do instrukcji EXTEST, z tą różnicą, że BS pozwala kontrolować i obserwować nie piny układu, ale wejścia i wyjścia logiki wewnętrznej
- RUNBIST, która wprowadza układ w tryb testowania samego siebie. Rezultaty tych testów można odczytać przy pomocy specjalnego rejestru podłączonego pomiędzy TDI i TDO
- HIGHZ, CLAMP ...

Standard pozwala również na wprowadzenie własnych instrukcji, odpowiednich dla danego typu układu, oraz dodatkowych rejestrów, służących niekoniecznie do celów testowych.

3. Po co tego używać

Test Access Port and Boundary Scan Architecture to narzędzie, które w ogromnym stopniu może uprościć testowanie elektroniki wyzwalania eksperymentów wysokiej energii:

1. Stanowi ono jednolite środowisko do przeprowadzania wielu rodzajów testów dotyczących różnych poziomów złożoności (modularności) systemu elektronicznego: od testowania pojedynczych układów scalonych poprzez aplikację wektorów testowych lub wykonywanie wbudowanych w układy testów własnych, do testowania płytek i połączeń między nimi. Może być ono również użyte do obserwacji sygnałów w trakcie normalnego działania elektroniki oraz przy jej setupie [25,Dodatek C], znacznie redukując ilość potrzebnych pinów. Uniwer-

salność tego narzędzia w tak zróżnicowanym, skomplikowanym i dużym systemie, jakim są systemy wyzwania, trudno przecenić.

2. Jest ono standardem przemysłowym. Układy zgodne z IEEE 1149.1 produkowane są przez główne firmy zajmujące się mikroelektroniką, jak Texas Instruments, Altera (np. urządzenia MAX i FLEX), Intel (m.in. mikroprocesory Pentium), Sun Microsystems (np. procesory UltraSPARC) i wiele innych. Przy projekcie układów ASIC dostępne są biblioteki definiujące podstawowe elementy Boundary-Scan.
3. Standard ściśle określa reguły projektowania układu z Boundary-Scan. Definiuje również język BSDL służący do opisu architektury Boundary-Scan w układzie. Umożliwia to częściową lub nawet pełną automatyzację zarówno przygotowania jak i przeprowadzenia testów przy pomocy komputera, korzystając ze standardowych formatów opisu budowy płytek i modułów [20,22,23,24]. Kilka firm elektronicznych (m.in. Teradyne, Corelis Inc., ASSET InterTech, JTAG Technologies B.V.) oferuje zestawy programów obsługujących i przygotowujących szereg testów, jak np. testy na zwarcia i złe połączenia (Virtual Interconnect Test), testy grup układów bez Boundary-Scan (Virtual Component and Cluster Test), testy logiki wewnętrznej układów (Boundary Functional Test) itp. Prace nad takim oprogramowaniem zostały rozpoczęte również w naszej grupie.
4. Stosowanie standardu IEEE 1149.1 ma wiele przewag nad użyciem metod ad hoc. Koszt implementacji Boundary-Scan jest dużo mniejszy, pozwala on na lepszą izolację błędów oraz przeprowadzanie testów niemożliwych do wykonania metodami ad hoc (np. możliwe jest wyprowadzenie wielu sygnałów testowych logiki wewnętrznej układu do rejestru brzegowego, bez przyłączania ich do pinów). Firma Texas Instruments przeprowadziła badania mające na celu porównanie metod ad hoc z testami z użyciem Boundary-Scan na przykładzie prostej płytki z pamięcią z dostępem VME [13]. Z badań tych wynikają następujące przewagi metod Boundary-Scan:
 - Lepsza obserwowalność i kontrolowalność
 - Lepsza detekcja i izolacja błędów
 - Większa niezawodność
 - Mniejsza liczba dodatkowych pinów
 - Mniejsza liczba części
 - Mniejsza liczba koniecznych testów

- Większa swoboda w kolejności przeprowadzanych testów
- Mniejsze zużycie energii

Użycie metod zgodnych z 1149.1 nie pozwoli jednak w zupełności zrezygnować z metod ad hoc. Standard ten jest trudny do wykorzystania w celu weryfikacji poprawności dynamicznego działania systemu ze względu na długi czas odczytu danych i przeszkody w synchronizacji mechanizmu Boundary-Scan z zegarem systemu. Kontrola przepływu danych przetwarzanych potokowo pomiędzy odległymi modułami tym bardziej wydaje się niemożliwa do realizacji.

V. Podsumowanie

W pracy położono nacisk na konieczność przeprowadzania dokładnych testów elektroniki wyzwiania oraz kontroli jej poprawnej pracy. Zaproponowano pewne systemowe metody budowy narzędzi testowych i kontrolnych ilustrując je przykładami z działającego detektora ZEUS. Opisano również narzędzia, które mogą w znacznym stopniu ułatwić testowanie aparatury w jej wszystkich fazach: od testowania prototypów w laboratorium, testów trygera w czasie przerw w działaniu detektora, aż po kontrolę pracy trygera w trakcie działania detektora.

Dodatek A. Testy w Bari

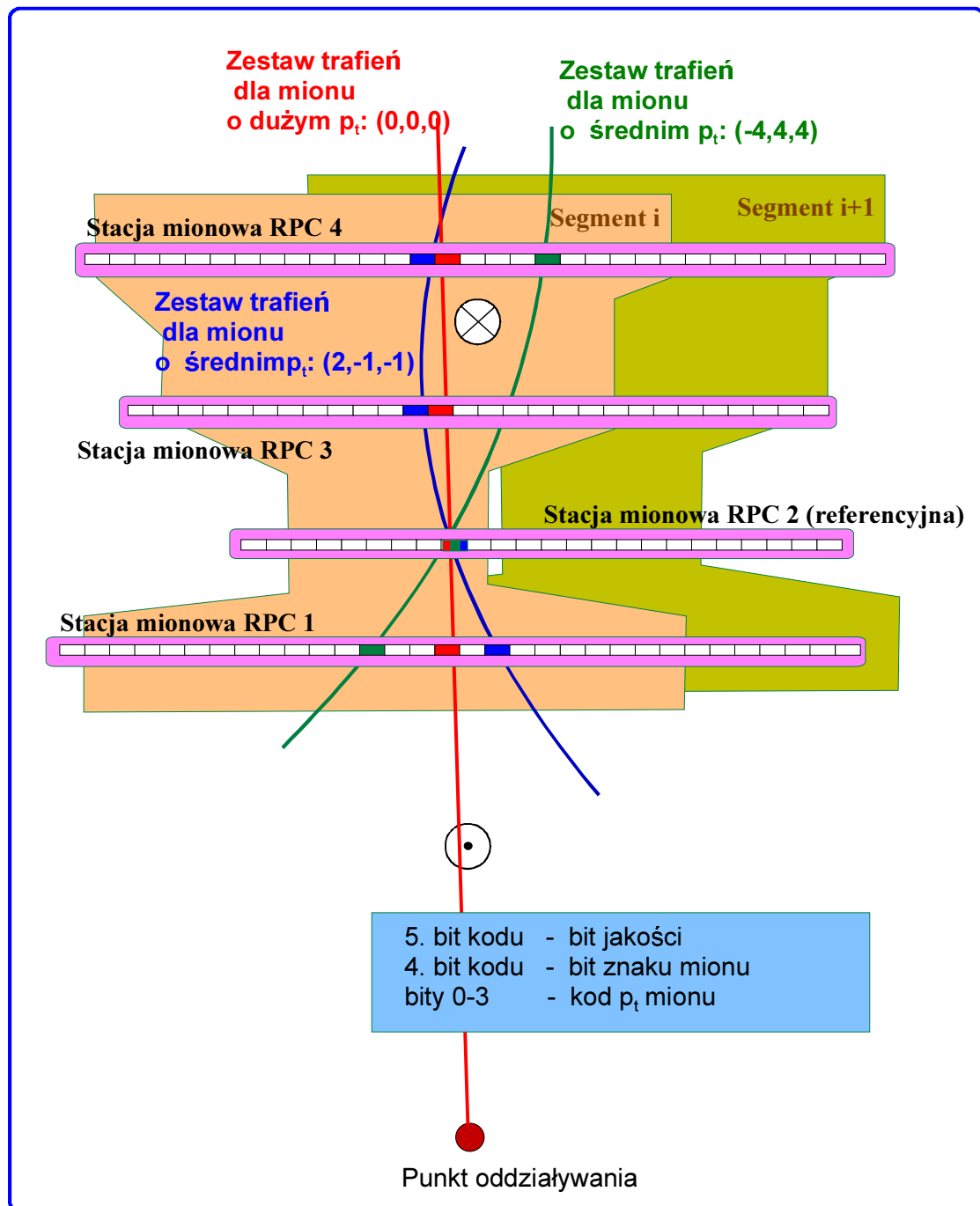
1. Wstęp

W październiku 1996 roku w Istituto Nazionale di Fisica Nucleare di Bari we Włoszech odbyły się testy prototypów pewnych części trygera RPC dla projektowanego detektora CMS. W niniejszym dodatku, po krótkim wprowadzeniu, zamieszczony został opis tych testów, jak również została podjęta próba ich podsumowania i wyciągnięcia z nich wniosków na przyszłość.

2. Algorytm działania trygera RPC

Głównym zadaniem trygera mionowego RPC jest identyfikacja i pomiar pędu mionu na podstawie śladu jaki zostawił on w czterech (lub trzech) warstwach komór RPC. Zakrzywienie toru jest funkcją pędu mionu. Z punktu widzenia trygera RPC przestrzeń detektora podzielona jest na tzw. segmenty o długości jednego paska i szerokości kilku pasków detektora RPC. Odpowiadają one obszarom o rozwartości 2.5 stopni w φ i 0.1 w η .

Każdy z segmentów analizowany jest przez oddzielny procesor PAC (Pattern Comparator). Po sczytaniu analogowych sygnałów z komór i konwersji na sygnały cyfrowe przez układy FEC (Rysunek 3. na stronie 5), synchronizacji, kompresji (LMUX), serializacji i nadaniu przez łącza optyczne, a następnie odebraniu sygnałów w pomieszczeniu pomiarowym i dekompresji (LDEMUX), dane z komór przekazywane są na płytę trygera, gdzie sygnał z czterech stacji każdego segmentu zostaje porównany przez układ PAC z wcześniej zdefiniowanymi zestawami sygnałów odpowiadającymi konkretnym wartościom pędu (Rysunek A.1). Z zestawów tych wybierany jest ten o najwyższym pędzie. Siedmiobitowa informacja o pędzie, o znaku mionu oraz o jakości pozostawionego śladu jest przekazywana do trygera wyższego stopnia i podlega dalszej analizie. Podczas analizy danych przez PAC i tryger wyższego stopnia informacja ta jest przechowywana w rejestrze przesuwym. Jeśli przypadek zostanie zaakceptowany, dane z rejestru przesuwego kopiowane są do bufora i czekają na odczyt przez system akwizycji danych.



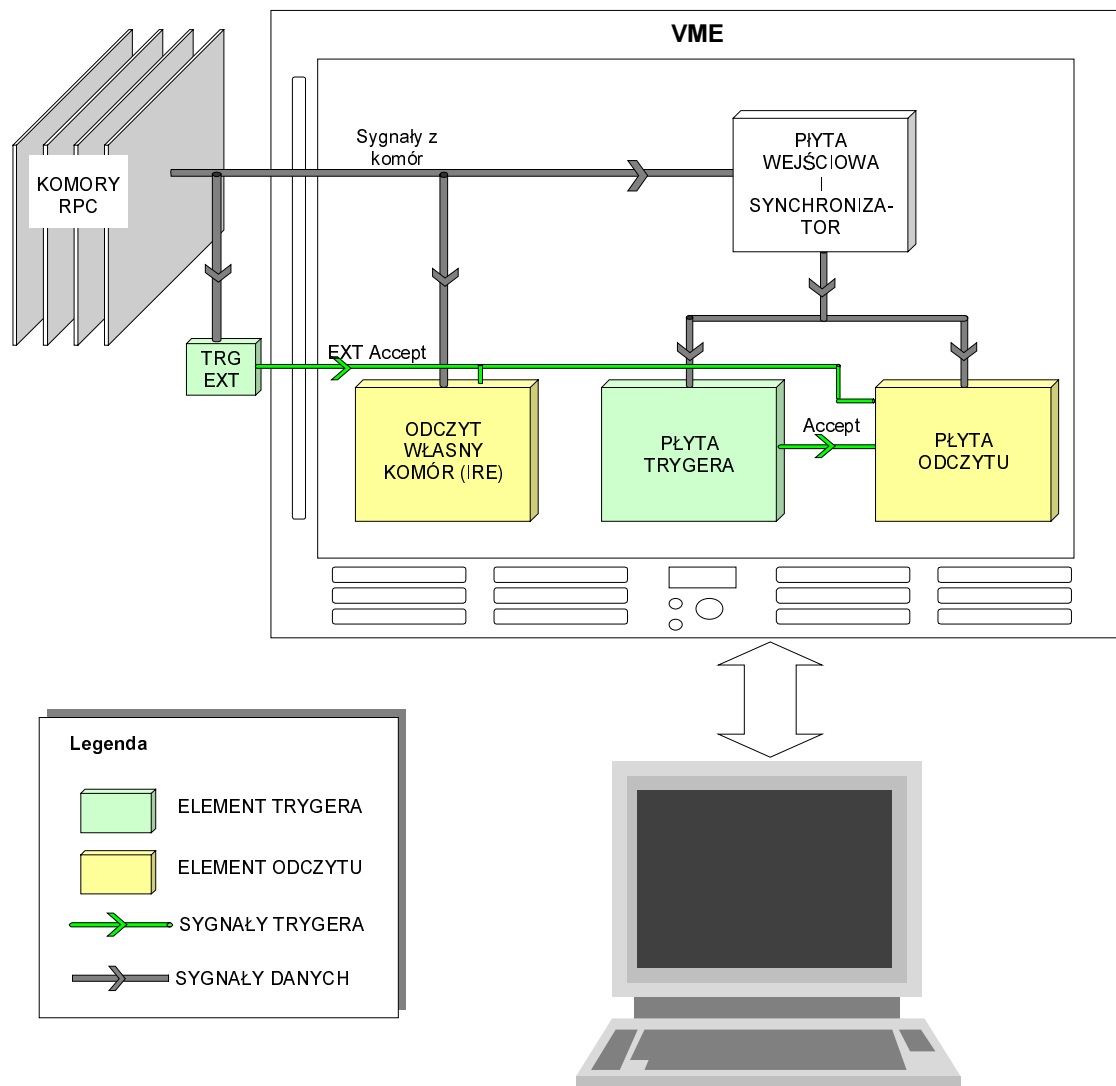
Ignacy Maciek Kudla, Warsaw University

Rysunek A.1. Przykłady śladów mionów i zestawów trafień

3. Stanowisko testowe

Stanowisko do testowania prototypu trygera podczas testów w Bari (Rysunek A.2) składało się z kilku płyt VME realizujących odbieranie sygnałów z komór, synchronizację, wykonanie algorytmu trygera i przygotowanie danych do odczytu przez komputer. Były to: płyty wej-

ściowe, płyta synchronizatora, płyta trygera i płyta odczytu. Kasetka VME połączona była ze stacją Sun za pomocą interfejsu Bit3.



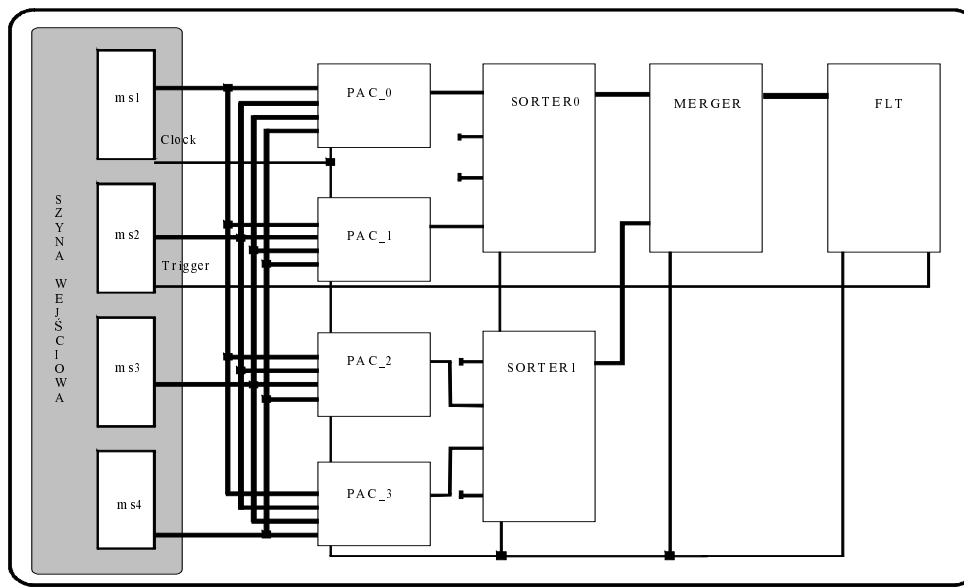
Rysunek A.2. Schemat stanowiska do testowania prototypu trygera podczas testów w Bari

Płyty wejściowe odbierały sygnały z komór lub generatora sygnałów FDPM i przekazywały je do płyty synchronizatora.

Płyta synchronizatora pobierała sygnały z płyty wejściowej. Jeśli mieściły się one w zaprogramowanym oknie, opóźniane były o odpowiednią liczbę taktów zegara, tak, by nastąpiła synchronizacja sygnałów z czterech płaszczyzn detektora. Następnie były one skracane i kierowane na szynę łączącą tę płytę z płytą trygera i płytą odczytu.

Płyta trygera sprawdzała, czy otrzymany od płyty synchronizatora układ trafień mógł zostać wygenerowany przez mion. Jeśli tak, to przypisywała mu ona pęd i gdy miał on wystarczają-

cą wartość wysyłała sygnał Accept do płyty odczytu. Płyta trygera składała się z czterech układów PAC, części sortującej i układu FLT (Rysunek A.3). Układy PAC sprawdzały, czy otrzymane zestawy trafień odpowiadały zaprogramowanym do procesorów PAC zestawom z przypisanymi im pięciobitowymi kodami pędów. Kody pędów były następnie sortowane przez 2 sortery i merger. Jeśli liczba kodów powyżej zdefiniowanego progu była wystarczająca, układ FLT wysyłał sygnał Accept do płyty odczytu.



Maciek Kudła, Warsaw University, Poland

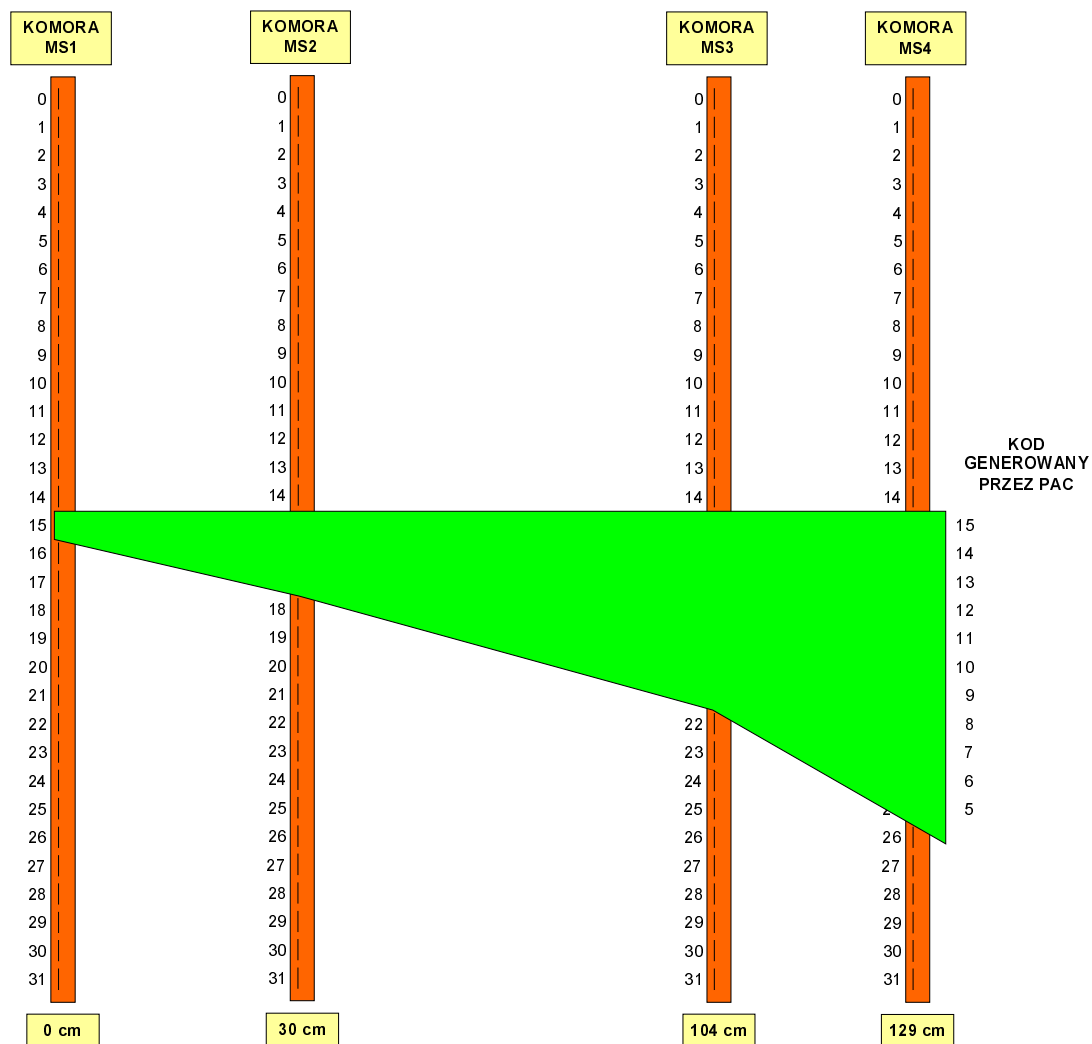
Rysunek A.3. Schemat płyty trygera

Płyta odczytu w każdym takcie zegara czytała dane z szyny i wprowadzała je do rejestru przesuwne (FIFO) o długości 128. Za każdym nadejściem sygnału Accept od płyty trygera, dane z wyjścia rejestru przesuwne (odpowiadające 3 taktom zegara) były kopiowane do bufora (derandomizer buffer). Następnie były one kompresowane, opatrzone numerem porządkowym i kopiowane do pamięci, która może pomieścić dane odpowiadające ośmiu sygnałom Accept. Stąd mogły być one przeczytane przez komputer za pośrednictwem kontrolera VME.

Większość kluczowych elementów tych płyt, a w szczególności procesory PAC, zrealizowane były przy pomocy programowalnych układów Altera. Fakt ten miał kluczowe znaczenie dla pomyślnego wyniku testów, co zostało wyjaśnione w kolejnych punktach dodatku.

Sygnały do wyżej opisanego prototypu trygera pobierane były z czterech płaszczyzn komór RPC ustawionych pionowo, równolegle, w odległościach 30, 104 i 129 cm od pierwszej z nich. Nie było żadnego sztucznego źródła cząstek, więc spodziewano się zarejestrować głównie promieniowanie kosmiczne. Nie było też dodatkowego pola magnetycznego: cząstki poruszały się po torach prostych. Kody generowane przez układy PAC, na podstawie których podejmowa-

na jest decyzja o zaakceptowaniu przypadku, zależały od kąta padania cząstki (Rysunek A.4).



Rysunek A.4. Kody generowane przez układy PAC w zależności od kąta padania cząstki

Każda z komór dawała po 32 sygnały odpowiadające pozycjom (paskom), przez które przeszła cząstka. Komory miały również swój własny, niezależny system trygerujący (EXTERNAL): liczono sumę logiczną sygnałów z każdej komory, a następnie wymagana była czterokrotna koincydencja sum logicznych. Były one także wyposażone w niezależny system akwizycji danych (IRE).

Ze względu na fakt, że natężenie promieniowania kosmicznego jest małe, a sygnały z komór były dość długie (600 ns) i charakteryzowały się dużymi rozrzutami czasowymi, synchronizator przy pracy z komorami nie spełniał dobrze swojej roli. Z tego powodu programy ładujące układy synchronizujące zostały przerobione w ten sposób, by sygnały z komór wchodziły na płytę trygera i odczytu bez zmian.

4. Przebieg i wyniki testów

Niniejsze testy są znakomitym przykładem problemów, jakie mogą pojawić się przy braku możliwości odpowiedniej kontroli poprawności funkcjonowania systemu, oraz braku narzędzi pozwalających na systemowe, automatyczne przetestowanie poszczególnych jego elementów. W pierwotnej wersji prototypu trygera nie było żadnej możliwości kontroli danych na poszczególnych etapach przetwarzania. Możliwe było jedynie podanie sygnałów testowych z generatora FDPM na synchronizator oraz odczyt danych z płyty odczytu.

Rezultatem tego było żmudne poszukiwanie przyczyn błędów w funkcjonowaniu systemu wszelkimi dostępnymi metodami oraz doklejanie do systemu narzędzi testowych, co i tak nie doprowadziło do wyjaśnienia wszystkich nieprawidłowości w działaniu trygera.

Przeprowadzone testy systemu podzielić można na dwie główne fazy:

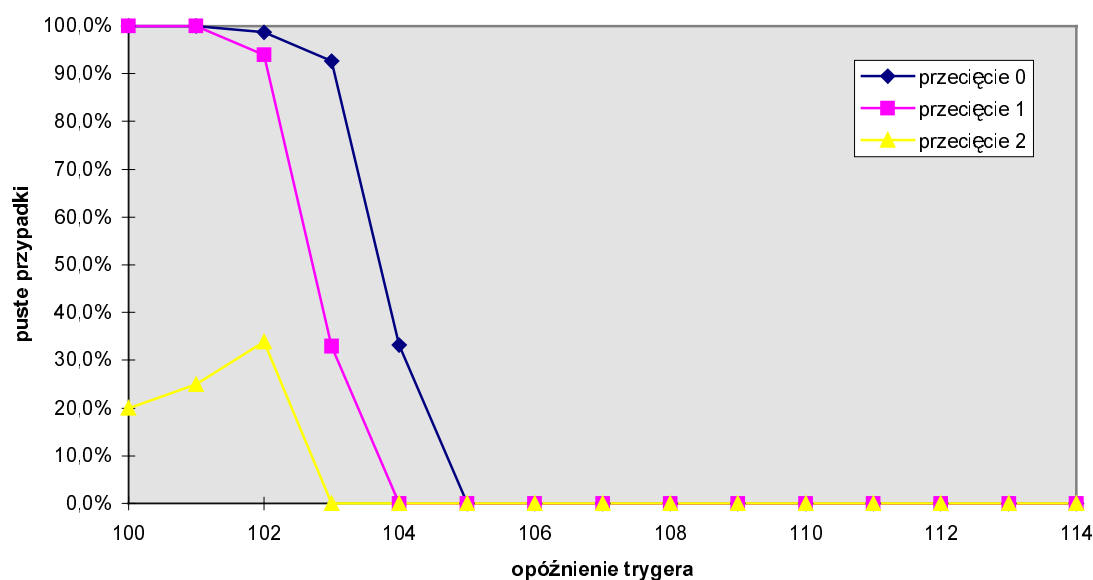
1. testy laboratoryjne: W fazie tej przeprowadzono wstępne testy ukierunkowane na weryfikację działania każdej płyty z użyciem generatora sygnałów FDPM. Ponieważ testy te nie były w pełni zautomatyzowane, płyta trygera przetestowana została jedynie poprzez podanie pobudeń odpowiadających wszystkim prostym oraz wybranym zakrzywionym śladom mionów. Testy laboratoryjne pomogły w znalezieniu błędów w działaniu sortera oraz uszkodzonych ścieżek na płycie trygera.
2. testy właściwe: Faza ta obejmuje testy prototypu przy pomocy sygnałów z komór RPC. Wykazała ona słabość narzędzi diagnostyczno-testowych wbudowanych w system. Błędy w działaniu trygera (na początku kilkanaście procent przypadków) usuwano stopniowo, poszukując ich przyczyn głównie za pomocą analizy danych wyjściowych. W późniejszym etapie dokonano zmian w systemie, przerabiając synchronizator w taki sposób, by po odpowiednim zaprogramowaniu symulował on prawdziwe sygnały z komór. To posunięcie niezmiernie ułatwiło proces testowania, w szczególności udało się odkryć uszkodzenia pinów w dwóch spośród czterech układów PAC. Jeden układ został wymieniony na nowy, w drugim natomiast został zmodyfikowany algorytm działania w ten sposób, by uszkodzony pin, odpowiadający paskowi nr 31 w stacji mionowej MS1 nie zaburzał pracy całego układu. Powyższy pasek przestał być widziany przez płytę trygera. Znaleziono także pewne błędy w oprogramowaniu (patrz pkt 5), które przez cały czas było udoskonalane.

Dane zbierano w trzech trybach pracy:

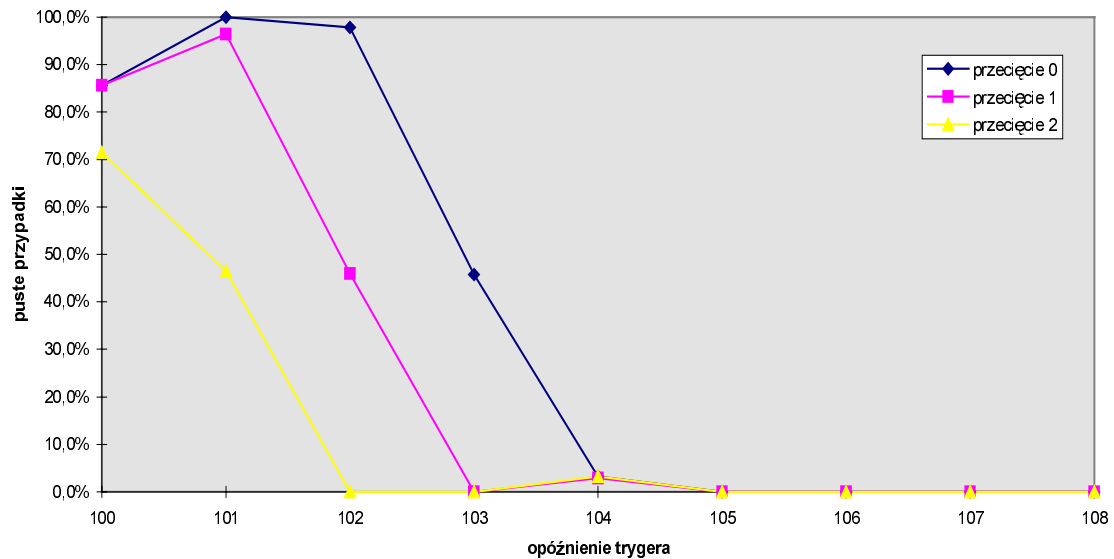
1. PAC: akceptacja przypadku na podstawie sygnału Accept z płyty trygera
2. EXT: akceptacja przypadku na podstawie trygera EXTERNAL

3. PAC+EXT: wymagana koincydencja sygnału Accept i trygera EXTERNAL

Przy pracy w trybie EXT, układy PAC nie generowały kodów. Dla każdego z trybów zebrano dane dla wielu wartości opóźnienia generowania sygnału Accept. Chodziło o to, by znaleźć optymalne opóźnienie, przy którym dane zebrane przez płytę odczytu najlepiej odpowiadają decyzji trygera. Dla zbyt krótkiego opóźnienia sygnał Accept był wystawiany zanim dane z rejestru przesuwnej płyty odczytu pojawiały się na wyjściu. Wtedy w przynajmniej jednym z trzech przecięć czytanych przez płytę odczytu nie pojawiały się żadne dane (lub dane nie były związane z decyzją trygera). Na Rysunkach A.5 i A.6 przedstawiono zależność liczby pustych przecięć od opóźnienia trygera. Dla opóźnień krótszych od 105 taktów zegara znaczna część czytanych danych jest pusta.

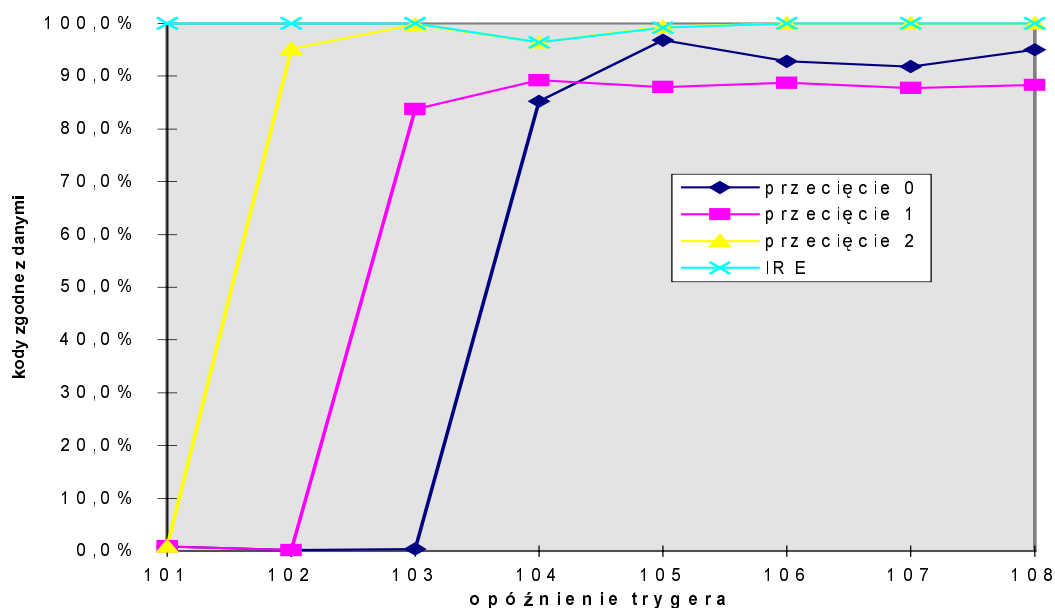


Rysunek A.5. Liczba pustych przypadków w zależności od opóźnienia trygera dla trybu PAC.



Rysunek A.6. Liczba pustych przypadków w zależności od opóźnienia trygera dla trybu PAC+EXT.

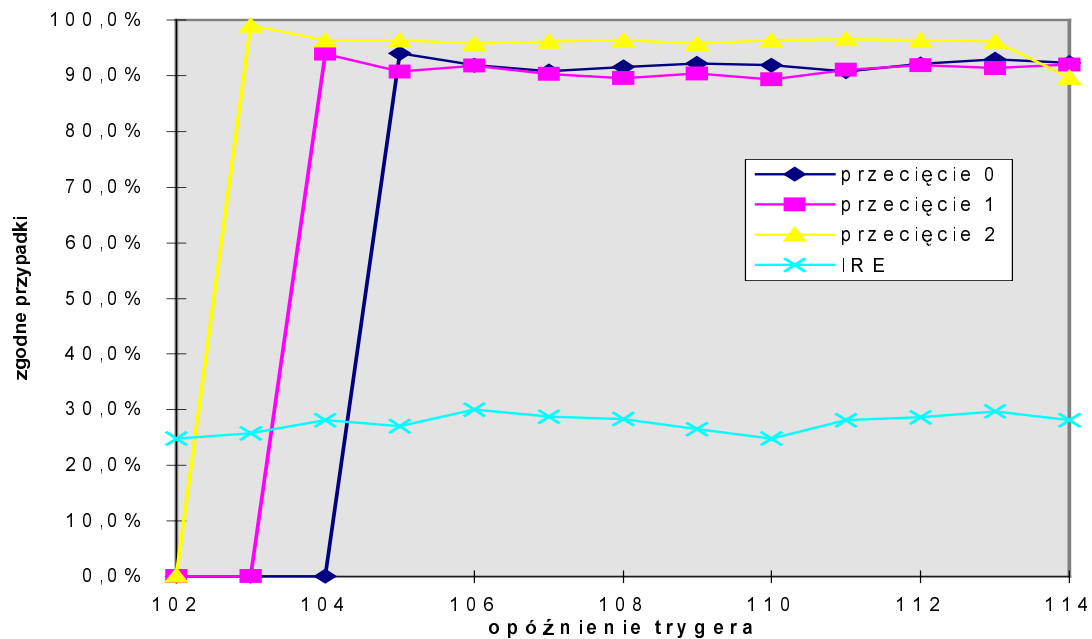
Poprawność kodów generowanych przez płytę trygera łatwo ocenić przyglądając się danym zebranym w trybie PAC+EXT (Rysunek A.7).



Rysunek A.7. Porównanie liczby przypadków z kodami PAC zgodnymi z danymi z płyty odczytu i z niezależnego systemu akwizycji danych IRE (tryb PAC+EXT).

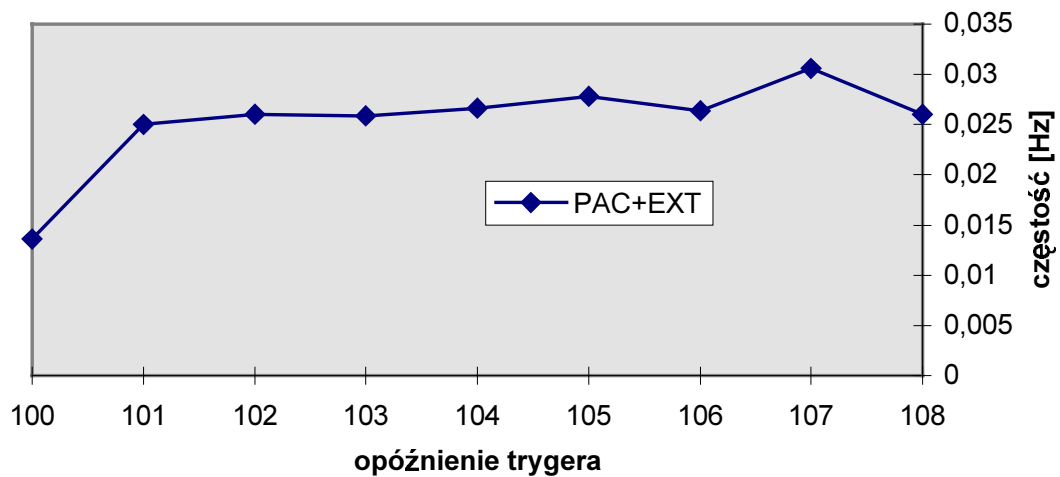
Dla wystarczająco długiego opóźnienia odpowiedź trygera jest w przybliżeniu w 100% zgodna z danymi z IRE i drugiego przecięcia z płyty odczytu. Niewielkie odstępstwa od 100% to przypadki, gdzie sygnał stanowi silny szum detektorów (kilkadziesiąt procent powierzchni detektora daje sygnał). Charakterystycznym jest fakt, że zgodność ta jest o kilka procent niższa dla danych z zerowego i pierwszego przecięcia. Okazuje się, że w tych przecięciach sygnały z pew-

nych pasków detektora są rzadziej rejestrowane (5. i 14. w MS2 i 20. w MS1 dla przecięcia nr 0, oraz 7. i 13. w MS1 dla przecięcia nr 1). Inaczej przedstawia się sytuacja dla trybu PAC (Rysunek A.8). Zgodność z danymi z IRE waha się w okolicach 27%. Mniejsza jest też zgodność z danymi z płyty odczytu z przecięcia nr 0 i 2, natomiast przecięcie nr 1 tym razem wypada korzystniej o ok. 3%.

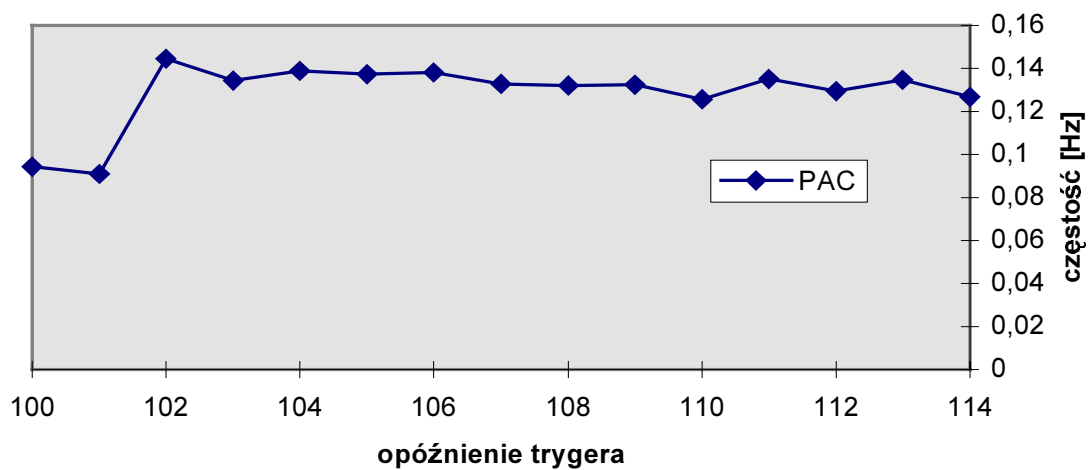


Rysunek A.8. Porównanie liczby przypadków z kodami PAC zgodnymi z danymi z płyty odczytu i z niezależnego systemu akwizycji danych IRE (tryb PAC).

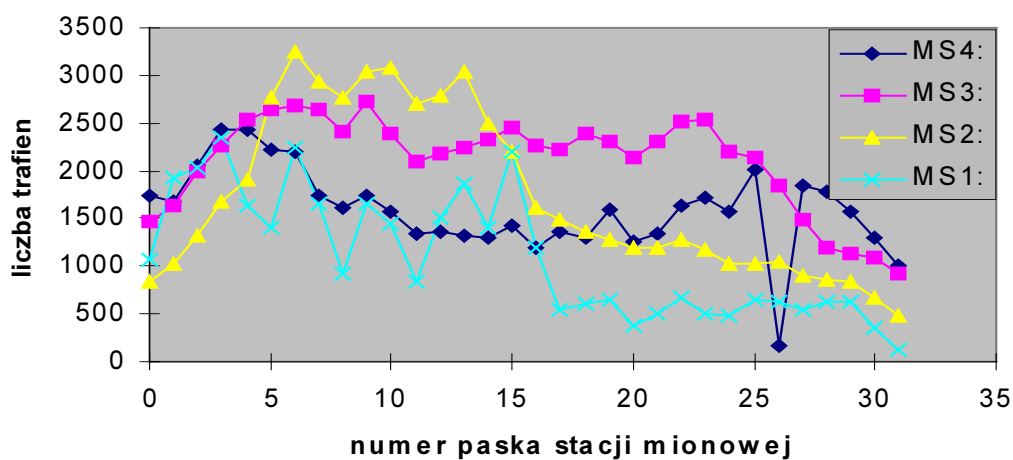
Innym interesującym faktem jest dużo mniejsza (kilkakrotnie) częstość zbierania danych w trybie PAC+EXT niż w trybie PAC (Rysunki A.9 i A.10). Ponadto w trybie PAC+EXT częstość ta wykazuje duże wahania. Porównując to z histogramem pokazującym częstości trafień w poszczególne paski detektora czytane za pomocą systemu IRE (Rysunek A.11), można domniemywać, że wkład do sumy logicznej przy trygerze EXTERNAL od połowy stacji MS1 i MS2 jest w pewnych okresach czasu bardzo zaniżony. Na wykresie tym widać również, że pasek 26. stacji MS1 jest prawie martwy.



Rysunek A.9. Zależność częstotliwości trygera od opóźnienia i od czasu (tryb PAC+EXT).



Rysunek A.10. Zależność częstotliwości trygera od opóźnienia i od czasu (tryb PAC).



Rysunek A.11. Liczby trafień w poszczególne paski stacji mionowych. (Dane odczytywane za pomocą systemu IRE, w trybie PAC oraz PAC+EXT)

5. Oprogramowanie

Przeprowadzenie testów byłoby niemożliwe bez współpracy z komputerem: część programowalnych układów scalonych ładowanych jest przez komputer, szereg parametrów płyt jest przez niego ustawianych itp. Konieczne stało się stworzenie specjalnego oprogramowania, które umożliwiłoby wygodne wykonanie tych czynności.

a) Opis działania

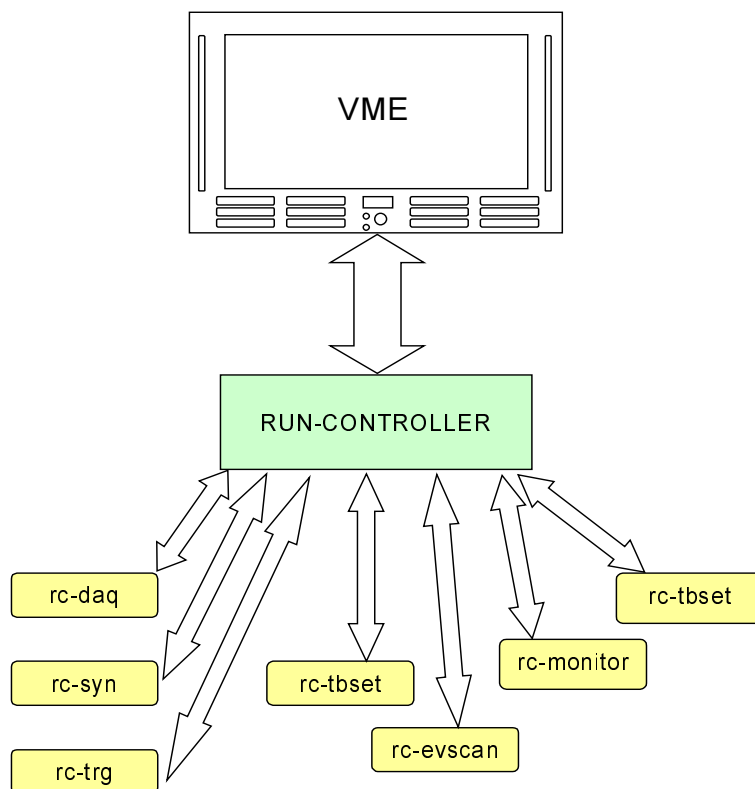
Planowany software musiał umożliwić:

- ładowanie programowalnych układów scalonych
- zapis i odczyt spod dowolnego adresu płyty VME
- ustawianie parametrów pracy płyt (opóźnień na synchronizatorze, progów akceptacji pędów na płycie trygera, trybów pracy płyt, itp.)
- odczyt danych z płyty trygera i płyty odczytu oraz ich zapis na dysk
- analizę zebranych danych zarówno w trakcie ich zbierania jak i później
- automatyczne tworzenie dokumentacji testów zawierającej wszystkie parametry pracy płyt, dokładny czas ich ustawienia itd.

Ze względu na szerokie wymagania zrezygnowano z idei stworzenia jednej lub kilku dużych, niezależnych aplikacji na rzecz mniejszych, odrębnych, współpracujących ze sobą programów. Takie podejście wydawało się prostsze w wykonaniu, pozwalało na łatwiejszą modyfikację programów oraz upraszczało znajdowanie błędów. Pojawił się jednak wtedy poważny problem: jak uniemożliwić dwóm programom działającym równolegle jednoczesny dostęp do VME? Dopuszczenie takiej możliwości wydawało się niebezpieczne, szczególnie w przypadku, gdy sposób adresowania którejs z płyt byłby odmienny (np. użycie innego modyfikatora adresu). Problem ten rozwiązano wykorzystując model klient-serwer. Jedynie serwer (nazwano go „run-controllerem”, RC) miałby możliwość bezpośrednich odwołań do VME. Programy-klienci odwołania do VME wykonywaliby za pośrednictwem run-controllera. Klienci mogliby też składać „zamówienia” na dostarczanie nadchodzących danych: w przypadku pojawienia się nowych danych na płycie odczytu, run-controller, po zweryfikowaniu ich poprawności, natychmiast przesyłałby je do klienta. Możliwa byłaby jednoczesna praca kilku klientów, ale tylko jednego serwera.

Zaimplementowany w opisany wyżej sposób system (Rysunek A.12) spełnia postawione mu wymagania. Składa się on z:

- *run-controllera*
- programów działających z *run-controller'em*
 - *rc-daq, rc-syn, rc-trg* (ustawianie i kontrola wybranych parametrów płyt)
 - *rc-tbset* (automatyczne ustawianie wszystkich parametrów płyt)
 - *rc-evscan* (wizualizacja danych on-line)
 - *rc-monitor* (oglądanie danych - tryb tekstowy)
 - *rc-pac* (wszechstronna analiza danych)
- programów działających bez *run-controllera*:
 - *vmeconf* (konfiguracja kasety VME)
 - *aboot, bootall* (ładowanie programowalnych układów scalonych)
 - *daq, syn, trg* (ustawianie i kontrola wybranych parametrów płyt)
 - *fdpm, mfdpm* (obsługa generatora impulsów FDPM)
 - *evscan, readout* (wizualizacja danych w trakcie ich zbierania)
 - *syn-test, syn-testx* (testowanie układów PAC przy pomocy płyty synchronizatora).



Rysunek A.12. Schemat komunikacji run controllera, klientów i VME.

Podczas pracy z *run-controller'em* wszystkie czynności są dokumentowane. *Run-controller* podczas uruchomienia tworzy katalog o nazwie odpowiadającej dacie i godzinie rozpoczęcia sesji. Do niego kopiowane są wszystkie pliki komunikacyjne zawierające daty i godziny poszczególnych odwołań do RC oraz dane przesyłane przez RC do klientów. Przykładowy fragment pliku z komendami wysyłanymi do run-controllera wygląda w sposób następujący:

```

% Run-Controller file : rclr.trc
% Generated on Mon Oct 28 19:20:10 1996
%
% User supplied comment: RC-Setup_ver.:1.0
1 19:20:10: CONNECT RC-Setup_ver.:1.0 *
2 19:20:11: SIGNAL TRG_RESET *
3 19:20:11: SET TRG_THRESHOLD_CODE0 1 b *
4 19:20:11: SET TRG_THRESHOLD_CODE1 1 b *
5 19:20:11: SET TRG_THRESHOLD_CODE2 1 b *
6 19:20:11: SET TRG_THRESHOLD_CODE3 1 b *
...

```

Do zalet systemu, poza automatyczną dokumentacją, należą: możliwość pracy bez RC (gdy nie ma groźby jednoczesnych odwołań do VME przez kilka programów, a zależy nam na większej prędkości działania), możliwość szybkiego ustawiania wszystkich parametrów pracy systemu (bootall, rc-tbset), możliwość pracy off-line z plikami komunikacyjnymi przy niewielkich modyfikacjach programów do pracy on-line.

b) Implementacja

System został napisany w językach C i C++. Długość kodu to w sumie ponad 15000 linii. Dzięki hierarchicznej budowie oraz zastosowaniu obiektowych technik programowania udało się osiągnąć dużą przejrzystość i łatwą modyfikowalność systemu.

Ważnym elementem systemu jest zestaw bibliotek, które dostarczają funkcje i klasy odpowiedzialne za określony fragment implementacji. Biblioteki te starają się ukryć przed użytkownikiem (np. projektantem programu-klienta) szereg niuansów związanych bezpośrednio z elektroniką, jak np. sposób komunikacji z VME, adresy płyt, adresy poszczególnych rejestrów, sposób buforowania danych przez płytę odczytu, metodę komunikacji klient-serwer itd. Zostaną one omówione pokrótce w następnym punkcie.

(1) Biblioteki

Na początku zostaną omówione biblioteki, które są bezpośrednio związane z urządzeniami elektronicznymi, na końcu zaś te, które dotyczą potencjalnego projektanta klienta.

Biblioteka `vme.h` jest odpowiedzialna za komunikację z VME. Dostarcza ona funkcji `VMEOpen()`, która otwiera deskryptor pliku związany z VME i ustawia odpowiednie parametry interfejsu Bit3. Funkcje `VMERead(...)` oraz `VMEWrite(...)` umożliwiają odczyt i zapis pod dowolny adres modułu VME. `VMEClose()` zamyka deskryptor pliku.

Biblioteki `trg-lib.hh` oraz `trg-def.h` przejmują na siebie wszystkie odwołania do płyty trygera. Zaimplementowana została w niej klasa `T_DAQ_TRG` oraz funkcje pozwalające zresetować płytę, ustawić lub odczytać parametry, w szczególności kody mionów:

```
enum TRG_SELECTOR {
    TRG_MODE = 0,
    TRG_THRESHOLD_CODE0,
    //...
    TRG_THRESHOLD_MULTIPLICITY,
    TRG_TRYGER_SOURCE,
    TRG_TRYGER_DELAY,
    TRG_TRYGER_REPETITION,
    TRG_TRYGER_COUNTER,
    TRG_INPUT_CODE0,
    //...
};

class T_TRG_BOARD {
    unsigned long address;
    int mode;
public:
    T_TRG_BOARD(unsigned long VME_addr);
    ~T_TRG_BOARD();
    MESSAGE Reset();
    MESSAGE Set(int selector, int parameter);
    MESSAGE Get(int selector, int *parameter);
    MESSAGE Get_all(TRG_PARAMETERS *parameters);
};
```

```
    MESSAGE Software_tryger_send();  
};
```

W podobny sposób zaimplementowane zostały biblioteki `daq-lib.hh`, `daq-def.h`, `syn-lib.hh` oraz `syn-def.h`.

W bibliotece `rc-prot.h` został zdefiniowany protokół komunikacyjny klient-serwer. Korzysta z niej `run-controller`, jak również klienci. W ten sposób zmiany w protokole (jak np. dodanie nowych komend, zmiany słów komunikacyjnych itp.) nie powodują nieporozumień pomiędzy klientem a serwerem.

Biblioteka `rc-int.h` zawiera deklaracje funkcji potrzebnych klientowi do komunikacji z serwerem (jest to jedyna potrzebna klientowi biblioteka). Są to między innymi funkcje:

```
MESSAGE rc_send(TCommand command, TSubcommand subcommand, void* param);  
MESSAGE rc_receive(TCommand *command, TSubcommand *subcommand, void **param);  
MESSAGE rc_offline(char *file_in, char *fileout);
```

Funkcja `rc_send(...)` służy do łączenia klienta z serwerem (wystarczy napisać `rc_send(RC_CONNECT, ...);`), wysłania żądania ustawienia wybranego parametru (np. `rc_send(RC_SET, RC_DAQ_MODE, RC_DAQ_MODE_SETUP);`), lub wysłania żądania odczytu nadchodzących przypadków (`rc_send(RC_MONITOR, RC_TB_EVENT, ...);`). Przysłane przez serwer przypadki odbiera się za pomocą funkcji `rc_receive(...)`. Możliwe jest również kontrolowanie stanu wybranych parametrów płyt: jeśli któryś z interesujących nas parametrów zostanie zmieniony, informację o tym dostaniemy również poprzez funkcję `rc_receive(...)`. Funkcja `rc_offline(...)` pozwala na pracę z plikiem komunikacyjnym bez rzeczywistej łączności z `run-controllerem`.

Biblioteka `tb.h` jest dołączana przez wszystkie pozostałe biblioteki. Jest w niej zdefiniowany m.in. typ `enum MESSAGE`, który określa najczęściej spotykane rodzaje błędów oraz funkcja `void tb_perror(char* s)` zwracająca słowny opis błędu, który ostatnio wystąpił.

(2) Run-controller

Mając niewiele czasu na napisanie całego systemu, zdecydowano się na użycie technik, które były dobrze znane autorom, rezygnując z tych, które wydawały się lepsze, lecz bardziej czasochłonne. Przede wszystkim do komunikacji klientów z serwerem użyto zwykłych plików, a nie, jak początkowo zakładano, 'Berkeley sockets'. Dzięki temu pozbyto się też problemu z automatycznym tworzeniem dokumentacji: pliki komunikacyjne tworzone w formacie ASCII, w łatwej do bezpośredniego zrozumienia przez człowieka formie stanowią wystarczającą dokumentację działań. Każdy klient potrzebuje czterech plików do komunikacji: dwa służą do wysy-

łania komend do run-controllera i odbierania od niego odpowiedzi (może to być informacja, czy komenda została zrealizowana, lub dane, o które poprosił klient) - są one wykorzystywane przez funkcję `rc_send(...)`, pozostałe dwa służą do tzw. „monitoringu” tj. przesyłania informacji o zmianach jakie wystąpiły w aparaturze (nadejście nowego przypadku, zmiana parametrów płyt ...) - używa ich funkcja `rc_receive(...)`.

Run-controller został napisany obiektowo. Poniżej omówione zostały zaimplementowane klasy.

Klasa `TFileHdl` umie otwierać i zamykać pliki komunikacyjne, ponadto przechowuje ich nazwy i katalogi, w których się znajdują.

Z klasy tej dziedziczą klasy `TReceiveFileHdl` i `TSendFileHdl`.

Klasa `TReceiveFileHdl` jest odpowiedzialna za odbiór komend przychodzących od klientów. Z każdym z klientów skojarzone są dwa obiekty tej klasy. Są tu zaimplementowane m.in. metody interpretujące polecenia dla run-controllera.

Klasa `TSendFileHdl` zajmuje się wysyłaniem danych do klientów. Z każdym z klientów skojarzone są również dwa obiekty tej klasy. Są tu zaimplementowane metody wysyłające dane i komendy do klientów.

Najważniejszą klasą jest `TComm`. Istnieje tylko jeden obiekt tej klasy. Do niej należy tworzenie obiektów klas `TReceiveFileHdl` i `TSendFileHdl`. Klasa ta zajmuje się też łącznością z VME poprzez obiekty klas `T_DAQ_BOARD`, `T_TRG_BOARD` i `T_SYN_BOARD`, zbiera dane, analizuje je i przesyła do klientów.

(3) Klienci

Większość z zaimplementowanych klientów run-controllera ma swoją wersję pracującą bezpośrednio z VME. System bibliotek został zaprojektowany w ten sposób, że decyzja o współpracy z run-controllerem może zostać podjęta nawet dopiero w momencie linkowania. Porównanie zbieranych danych przez dwie wersje tego samego programu (jedna korzystająca z run-controllera, druga odwołująca się bezpośrednio do VME) ułatwiło odnalezienie kilku błędów w run-controllerze.

6. Konkluzje

Testy prototypu trygera mionowego RPC pokazały, że nawet w przypadku niewielkiego systemu składającego się z zaledwie kilku płyt elektronicznych należy być przygotowanym na występowanie wielu różnego rodzaju błędów. Problem oczywiście będzie dużo poważniejszy,

gdy będziemy mieli do czynienia z docelowym systemem. Odkrycie podobnych błędów przy braku zaawansowanych metod testowych w docelowym systemie byłoby zadaniem niezmiernie trudnym, jeśli nie niemożliwym. Powyższe testy nasuwają pewne sugestie.

Po pierwsze, większość błędów związana była z mechanicznymi uszkodzeniami płytek lub układów scalonych. Objawiało się to w złym przesyłaniu sygnałów. Sugeruje to zatem potrzebę zastosowania pewnych narzędzi do automatycznego testowania połączeń na płytkach. Ze względu na zwiększającą się miniaturyzację, jak również powszechne stosowanie montażu powierzchniowego wydaje się, że odpowiednim do tego narzędziem byłby Boundary-Scan. Przy jego użyciu błędy związane z uszkodzeniami ścieżek lub pinów układów można wyizolować w bardzo krótkim czasie i przy niewielkim wkładzie pracy.

Po drugie, niezbędne wydaje się wbudowanie w system urządzeń testowych ad-hoc. W razie problemów z synchronizacją czy też z innymi dynamicznymi aspektami systemu, zastosowanie mechanizmu Boundary-Scan niewiele by pomogło. Dobrze by było, gdyby istniała możliwość odczytu danych lub pewnych wartości kontrolnych (np. liczników, sum sygnałów itp.), oraz możliwość pobudzania elektroniki dowolnymi sygnałami na pośrednich etapach przetwarzania.

Po trzecie, pomocne może być użycie programowalnych układów scalonych, takich jak urządzenia Altera FLEX. Algorytm ich działania może być w każdej chwili zmieniony, co jest szczególnie użyteczne w fazie projektowania systemu. Również w systemie docelowym część układów mogłaby być programowalna tak, by w okresach przerw w eksperymencie mogły one spełniać funkcje testowe podobnie jak opisany wcześniej synchronizator.

Dodatek B. Programy do obsługi FDPM

1. Wstęp

FDPM jest płytą VME, która może transmitować lub przyjmować dane z częstotliwością do 100 MHz przez cztery ośmiobitowe kanały, co daje przepustowość do 400 MB/sek (patrz Rysunek 4 na stronie nr 11). Zawiera ona pół megabajta pamięci, układ scalony VLSI MAG (Memory Address Generator) [9], oraz szereg rejestrów. Szybkie wejścia/wyjścia połączone są równolegle z czterema blokami pamięci. Układ MAG w inteligentny sposób kontroluje adresowanie bloków pamięci. Bloki pamięci mogą być dostępne przez VME lub używane podczas transmisji sygnałów. Fizycznie każdy blok jest zaimplementowany w postaci czterech multipleksowanych elementów pamięci. W ten sposób prędkość przesyłania danych może być cztery razy szybsza niż prędkość dostępu do pamięci. Cztery bajty mogą być równolegle transmitowane lub przyjmowane z częstotliwością określoną przez zewnętrzny zegar (do 100 MHz). Rejestry są częścią programowalnej logiki kontrolnej. Określa się w nich m.in. adresy i długości rejonów pamięci, z których się chce korzystać. Rejony te mogą być wysyłane lub zapisywane po kolei lub w pętli. Zwiększa to znacząco długość możliwych do wygenerowania sekwencji sygnałów.

W celu łatwej obsługi FDPM zostało w naszej grupie napisane oprogramowanie składające się z biblioteki C++ oraz kilku programów użytkowych, które pozwala na zaprogramowanie i kontrolę FDPM w celu wysyłania sekwencji sygnałów. Zostały one opisane w tym dodatku.

2. Składnia skryptów *fsc*

Dwa podstawowe programy do obsługi FDPM: *nfdpm* oraz *xfdpm*, operują na skryptach *fsc* (fdpm script). Skrypty te, korzystając z rozbudowanej składni, w hierarchiczny sposób definiują stan wszystkich 32 bitów sygnałów wyjściowych dla każdego taktu zegara.

W celu precyzyjnego opisu składni skryptów w dalszej części opisu użyto następujących pojęć:

- kanal*: jeden bit (druć) sygnału wyjściowego; każdy kanał posiada numer identyfikacyjny w granicach od 0 do 31
- sygnał*: grupa kilku kanałów zazwyczaj spełniająca pewną określoną funkcję
- sekwencja*: seria stanów wszystkich kanałów dla jednego lub więcej taktów zegara; odpowiada jej pewien określony obszar pamięci FDPM zawierający definicję stanów wszystkich kanałów dla kolejnych taktów zegara. Adresy rejonów pamięci przeznaczonych

nych do wysłania określone są w rejestrach FDPM

Każdy skrypt *jsc* składa się z dwóch części: sekcji definicji sekwencji oraz sekcji układu sekwencji. W pierwszej z nich definiuje się rodzaje sekwencji, z których się chce korzystać. Część ta opisuje, w jaki sposób ma zostać zaprogramowana pamięć FDPM. Sekcja układu sekwencji określa natomiast kolejność oraz krotność wysyłania zdefiniowanych w pierwszej części sekwencji. Część ta opisuje sposób zaprogramowania rejestrów.

Składnia skryptów *jsc* została przedstawiona poniżej za pomocą kilku przykładów.

Przykład 1. Wygenerowanie sekwencji bitów 0110 na wszystkich trzydziestu dwóch kanałach FDPM zrealizowane jest przez poniższy skrypt.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tu zaczyna się sekcja definicji sekwencji %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DEFSIG all      %definicja sygnału o nazwie 'all'
  31 30 29 28 27 26 25 24
  23 22 21 20 19 18 17 16
  15 14 13 12 11 10  9  8
   7  6  5  4  3  2  1  0
ENDSIG

DEFSEQ my_seq   %definicja sekwencji o nazwie 'my_seq'
  SET all 0
  STEPS 1
  SET all &hFFFFFFF
  STEPS 2
  SET all 0
  STEPS 1
ENDSEQ

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tu zaczyna się sekcja układu sekwencji %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SENDSEQ my_seq 1  %jednokrotne wysłanie sekwencji 'my_seq'
```

Skrypt został opatrzony komentarzami, które rozpoczyna znak %. Komentarz rozciąga się do końca linii. Na początku skryptu znajduje się sekcja definicji sekwencji. Aby zdefiniować sekwencję najpierw musi zostać zdefiniowany sygnał (DEFSIG ... ENDSIG) grupujący kanały, których stany mają być pod kontrolą. Tutaj sygnał składa się ze wszystkich trzydziestu dwóch kanałów i została mu nadana nazwa 'all'.

Następnie definiowana jest sekwencja (DEFSEQ ... ENDSEQ) o nazwie 'my_seq' ustawiająca sygnał 'all' (tzn. wszystkie kanały) kolejno do stanów 0110. Dzieje się to w kilku krokach. Najpierw określany jest stan wszystkich kanałów przy użyciu komendy 'SET *nazwa_sygnału* *liczba*'. Kolejne bity *liczby* określają stan kolejnych kanałów sygnału. *Liczba* może być zapisana w

jednym z trzech formatów: dziesiętnym, heksadecymalnym lub binarnym. Przy zapisie heksadecymalnym *liczba* musi być poprzedzona prefiksem &h, a przy zapisie binarnym prefiksem &b.

Po określeniu stanu kanałów należy podać ilość taktów zegara dla których taki stan kanałów ma się utrzymywać (instrukcja 'STEPS *liczba*').

W zdefiniowanej powyżej sekwencji 'my_seq' najpierw ustawiane są wszystkie kanały do stanu 0 ('SET all 0'). Taki stan ma się utrzymywać przez jeden takt zegara ('STEPS 1'). Następnie wszystkie kanały ustawiane są do stanu 1 ('SET all &hFFFFFF') i utrzymywane w tym stanie przez dwa takty zegara ('STEPS 2'). Następnie ponownie kanały ustawiane są do stanu 0 i trwa to przez jeden takt zegara.

Sekcja układu sekwencji znajduje się na końcu skryptu. Instrukcja 'SENDSEQ my_seq 1' powoduje takie zaprogramowanie odpowiednich rejestrów FDPM, by sekwencja my_seq została wysłana jednokrotnie.

Przykład 2. Wygenerowanie sekwencji imitujących przejście mionów przez cztery warstwy komór. Każda z komór składa się z ośmiu pasków. Dane imitują jeden takt zegara bez obecności mionów, a następnie przejście dwóch mionów: jednego prostopadłego do płaszczyzny komór i jednego ukośnego.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tu zaczyna się sekcja definicji sekwencji %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%definicje sygnalow z kolejnych plaszczyzn komor:
DEFSIG ms1  31 30 29 28 27 26 25 24  ENDSIG
DEFSIG ms2  23 22 21 20 19 18 17 16  ENDSIG
DEFSIG ms3  15 14 13 12 11 10  9  8  ENDSIG
DEFSIG ms4   7  6  5  4  3  2  1  0  ENDSIG

%definicja makra 'pusto' - brak mionu
DEFM pusto
    SET ms1 0
    SET ms2 0
    SET ms3 0
    SET ms4 0
    STEPS 1
ENDM

%definicja makra 'prosty' - przejście mionu
% prostopadłego do plaszczyzn komor
DEFM prosty
    SET ms1 &b00010000
    SET ms2 &b00010000
    SET ms3 &b00010000
    SET ms4 &b00010000
    STEPS1
ENDM

%definicja makra 'ukosny' - przejście mionu ukosnego

```

```

DEFM ukosny
    SET ms1 &b01000000
    SET ms2 &b00100000
    SET ms3 &b00010000
    SET ms4 &b00001000
    STEPS 1
ENDM

%definicja sekwencji 'my_seq' przy użyciu makr
DEFSEQ my_seq
    pust0
    prosty
    ukosny
ENDSEQ

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tu zaczyna się sekcja układu sekwencji %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SENDSEQ my_seq 1

```

W tym przykładzie wygodnie było zdefiniować cztery sygnały: po jednym sygnale dla każdej płaszczyzny komór. Użyto tu makr reprezentujących brak mionu, przejście prostopadłego mionu i przejście ukośnego mionu. Istnieją dwie podstawowe różnice pomiędzy makrem a sekwencją.

Po pierwsze, sekwencja zajmuje pewien określony obszar pamięci FDPM. W skrypcie *fs* musi być zdefiniowana zawsze przynajmniej jedna sekwencja. Makro natomiast jest obiektem pomocniczym, 'klockiem', z którego można składać kolejne makra lub sekwencje. Makro nie ma przypisanego dla siebie fragmentu pamięci FDPM.

Po drugie, sekwencja, ze względu na budowę FDPM, ma zawsze długość będącą wielokrotnością liczby 16. Jeśli definicja sekwencji w skrypcie nie spełnia tego warunku, sekwencja jest sztucznie wydłużana poprzez dopisanie zer na końcu. Makro ma natomiast długość określoną w skrypcie.

Przykład 3. Przykład ten jest bardziej rozbudowaną wersją przykładu drugiego. Chcemy wysłać wielokrotnie powtarzające się pewne układy mionów np. w celu sprawdzenia czy aparatura wykrywająca miony działa powtarzalnie. Najpierw wysyłana jest tysiąc razy sekwencja symulująca przejście dziesięciu prostych i piętnastu ukośnych mionów, a następnie również tysiąc razy sekwencja symulująca następujących po sobie dwadzieścia pięć prostych i dwadzieścia ukośnych mionów.

```

DEFSIG ms1 31 30 29 28 27 26 25 24 ENDSIG
DEFSIG ms2 23 22 21 20 19 18 17 16 ENDSIG
DEFSIG ms3 15 14 13 12 11 10 9 8 ENDSIG

```

```

DEFSIG ms4   7   6   5   4   3   2   1   0   ENDSIG

DEFM pusto
  SET ms1 0
  SET ms2 0
  SET ms3 0
  SET ms4 0
  STEPS 1
ENDM

DEFM prosty
  SET ms1 &b00010000
  SET ms2 &b00010000
  SET ms3 &b00010000
  SET ms4 &b00010000
  STEPS1
ENDM

DEFM ukosny
  SET ms1 &b01000000
  SET ms2 &b00100000
  SET ms3 &b00010000
  SET ms4 &b00001000
  STEPS 1
ENDM

DEFSEQ my_seq1
  pusto
  REPEAT 10  %dziesięciokrotne wyslanie prostego mionu
  prosty
  ENDR
  REPEAT 15  %pietnastokrotne wyslanie ukosnego mionu
  ukosny
  ENDR
ENDSEQ

DEFSEQ my_seq2
  pusto
  REPEAT 25
    prosty
  ENDR
  REPEAT 20
    ukosny
  ENDR
ENDSEQ

SENDSEQ my_seq1 1000  %tysiackrotne wyslanie sekwencji 'my_seq1'
SENDSEQ my_seq2 1000  %tysiackrotne wyslanie sekwencji 'my_seq2'

```

Nowym elementem w tym przykładzie jest zastosowanie pętli REPEAT dla wielokrotnego wysłania prostych i ukośnych mionów. Zwiększa to oczywiście rozmiar sekwencji. Tysiąckrotne wysłanie sekwencji 'my_seq1' oraz 'my_seq2' realizowane jest przez odpowiednie zaprogramowanie rejestrów (instrukcja SENDSEQ), przez co ilość użytej pamięci FDPM się nie zmienia.

Innym rozwiązaniem tego problemu mogłoby być tysiąckrotne wydłużenie samych sekwencji, np. przez zastosowanie zewnętrznej pętli REPEAT:

```

DEFSEQ my_seq1
  REPEAT 1000
    pusto

```

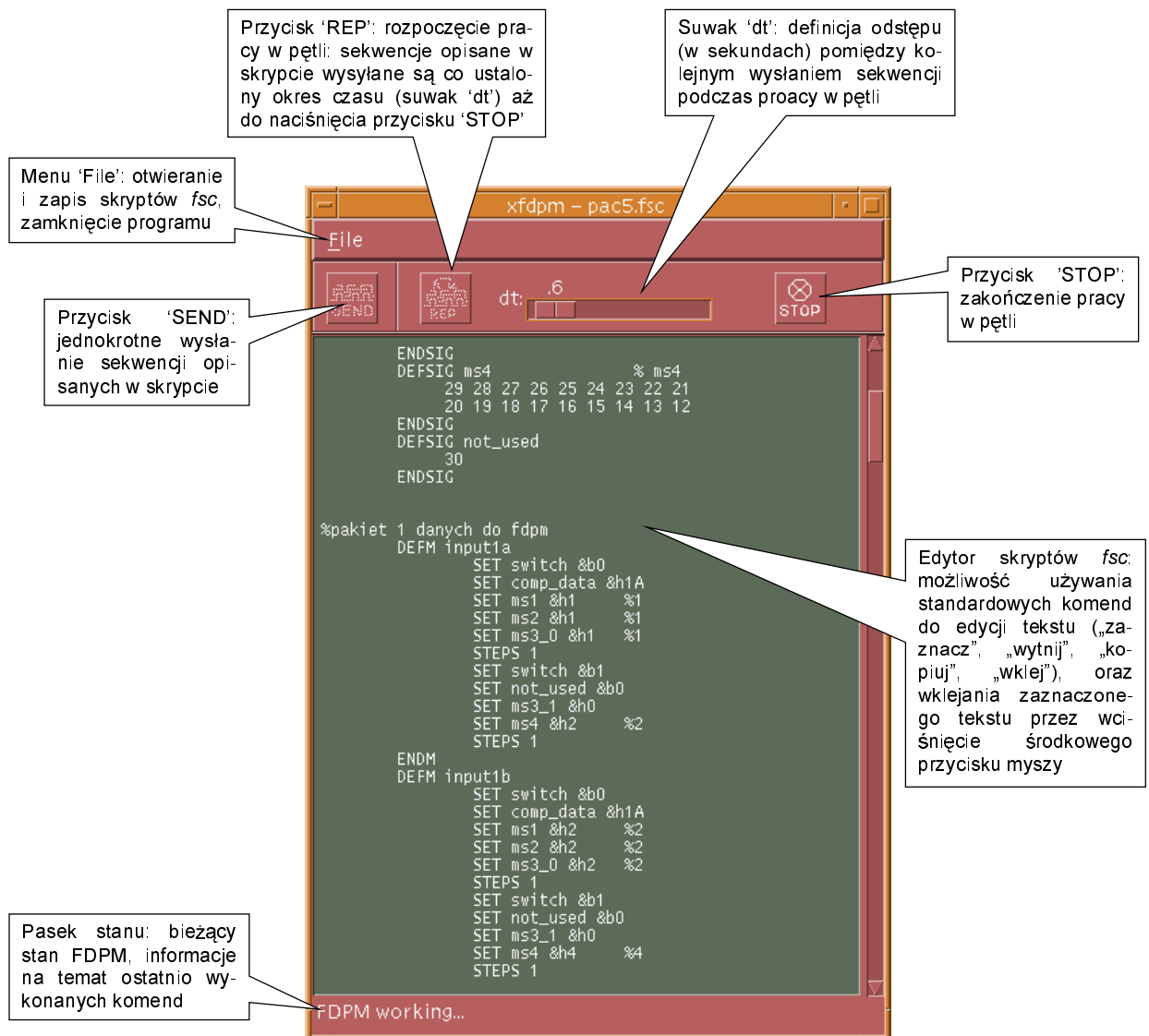
```
REPEAT 10  %dziesięciokrotne wysłanie prostego mionu
    prosty
ENDR
REPEAT 15  %piętnastokrotne wysłanie ukosnego mionu
    ukosny
ENDR
ENDR
ENDSEQ
```

Zaletą takiego rozwiązania jest częściowe ominięcie wspomnianego faktu dotyczącego automatycznego wydłużania długości sekwencji do wielokrotności liczby 16 – pomiędzy kolejnymi powtórzeniami wysłania dziesięciu prostych i piętnastu ukosnych mionów będzie dokładnie jeden pusty takt zegara. Wadą takiego podejścia jest duże zużycie pamięci FDPM.

3. Przegląd programów

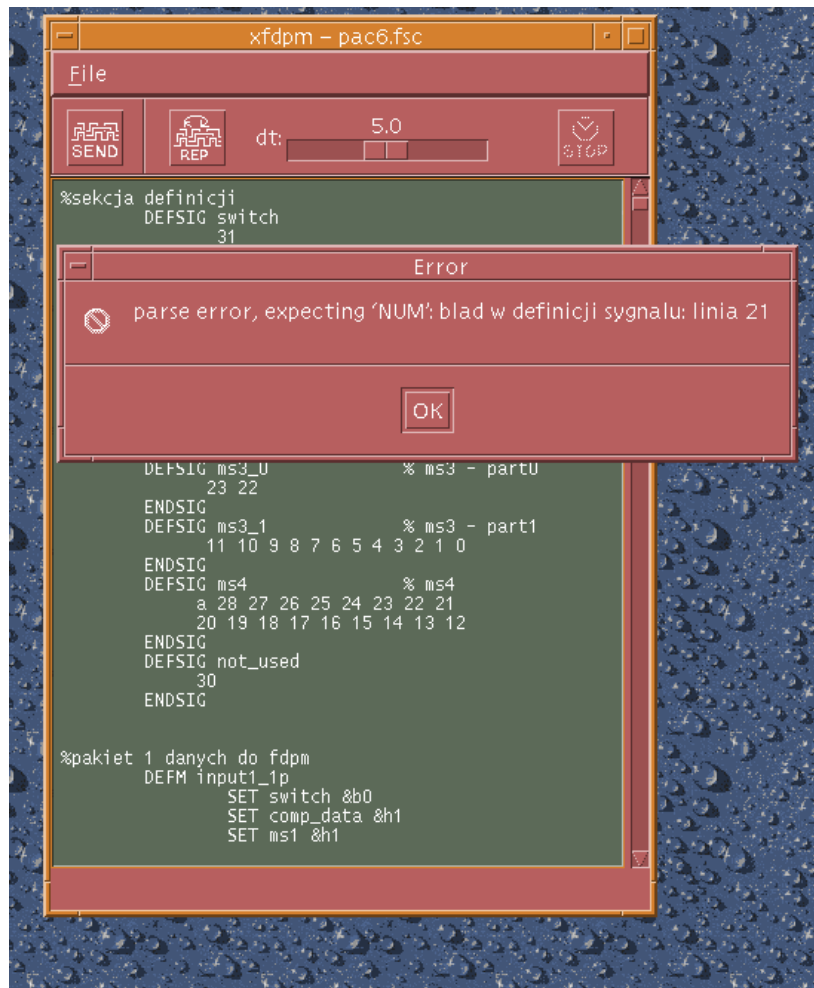
Jak już zostało wspomniane powyżej, obydwie napisane w naszej grupie programy do obsługi FDPM: *nfdpm* oraz *xfdp* operują na skryptach *fsc*. *Nfdpm* jest tekstowym interfejsem, który potrafi interpretować żądany skrypt, odpowiednio zaprogramować i uruchomić FDPM.

Xfdpm jest graficznym środowiskiem (Motif, CDE) do obsługi FDPM. Posiada on wbudowany edytor skryptów *fsc* oraz wygodny w użyciu interfejs użytkownika pozwalający na wielokrotną interpretację skryptu zawartego w edytorze, załadowanie i uruchomienie FDPM, oraz pracę ciągłą, podczas której FDPM wysyła dane co ustalony okres czasu (Rysunek B.1).



Rysunek B.1. Główne okno programu *xfdpm* podczas pracy w pętli

Obydwa programy korzystają z tej samej funkcji interpretującej skrypty *fsc*, która w razie wystąpienia błędu składniowego w skrypcie podaje numer błędnej linii oraz rodzaj błędu (Rysunek B.2).



Rysunek B.2. Informacja o błędzie składniowym w skrypcie *fsc*. W definicji sygnału *ms4* zamiast liczby znajduje się litera „a”.

4. Implementacja

Obydwa programy zaimplementowane zostały w językach C oraz C++ na stacji roboczej Sun.

Jako niskiego rzędu interfejs do FDPM utworzono grupę klas reprezentujących pamięć, procesor MAG (Memory Address Generator) wraz z rejestrami FDPM, oraz sekwencję. Zaprogramowanie FDPM polega na utworzeniu jednej lub kilku sekwencji (czyli obiektów określonej klasy), które chcemy wysłać, „wstawieniu” ich do pamięci korzystając z odpowiedniej metody, oraz zaprogramowaniu rejestrów poprzez „wstawienie” do reprezentującego je obiektu handlera do wpisanej do pamięci sekwencji. Biblioteka ta może być wykorzystana przez dowolne programy napisane w C lub C++ (dla przykładu patrz Dodatek D).

Interpreter skryptów *fsc* stanowi oddzielną funkcję korzystającą z wyżej opisanej bibliote-

ki. Został on wygenerowany z użyciem generatora analizatorów składniowych *bison*, oraz analizatora leksykalnego *flex*.

Interfejs graficzny do programu *xfdpm* zaprojektowany został za pomocą programu *CDE Application Builder*.

Dodatek C. JTAG software

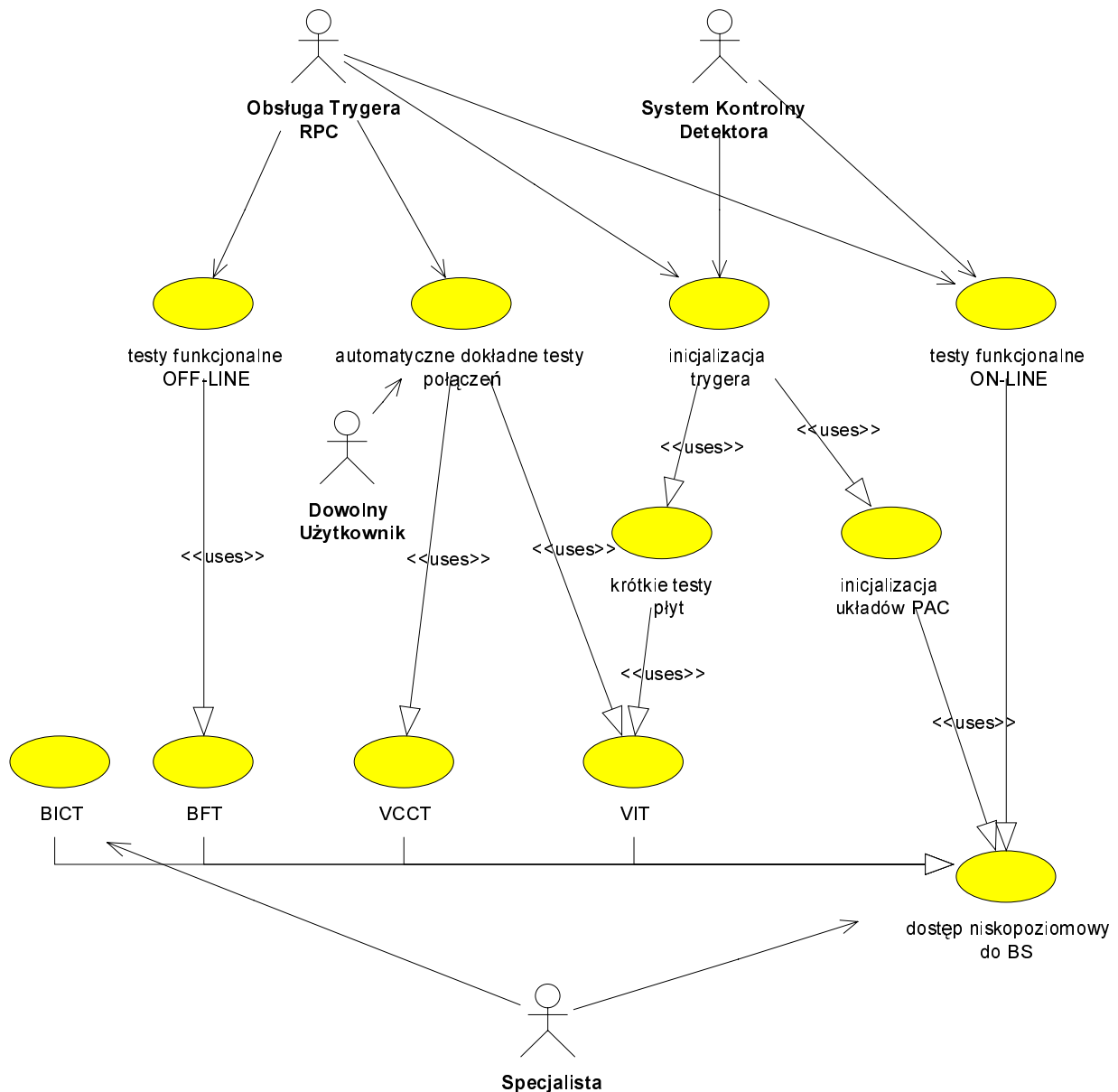
5. Wprowadzenie

Dodatek ten poświęcony został omówieniu projektowanego w naszej grupie systemu obsługującego standard IEEE 1149.1 (Test Access Port and Boundary Scan Architecture [15]). Prace te związane są z wykorzystaniem standardu przy budowie trygera RPC dla detektora CMS [5,6], a w szczególności jego integralnej części: procesora PAC (patrz Dodatek D). Ustawienie tego procesora w żądany tryb pracy następować będzie przy użyciu mechanizmów BS (specjalnych instrukcji BS - USER_7, USER_8, USER_9, USER_A, USER_B – dodanych w tym celu zgodnie z ustaleniami standardu). Planowane jest także użycie standardu w celach testowych. Aby w pełni wykorzystać możliwości płyt z BS, należy posiadać w tym celu programy, które mogą w pełni zautomatyzować przygotowanie oraz wykonanie szeregu niezbędnych testów. Celem dodatku jest zebranie różnych spostrzeżeń na temat systemu oraz krótka prezentacja wykonanych dotychczas prac.

Zadania wymagane od systemu zostały przedstawione na Rysunku C.1 razem z przypisaniem ich aktorom (czyli użytkownikom lub systemom, na których żądanie zadania te będą wykonywane). Ten, jak również większość innych zamieszczonych w tym dodatku diagramów korzysta z powszechnie przyjętej notacji UML (*Unified Modeling Language* [26,27]).

Aktorzy występujący w systemie to:

- *System Kontrolny Detektora*: interfejs do programów kontrolujących pracę detektora CMS (Run Controllera lub jego odpowiednika)
- *Obsługa Trygera RPC*: osoba zajmująca się poprawnym działaniem trygera RPC, niezbyt wtajemniczona w szczegóły techniczne trygera
- *Specjalista*: osoba znająca szczegóły techniczne trygera, uczestnicząca w jego projektowaniu. Pewne elementy systemu są używane przez nią również w fazie projektowania.
- *Dowolny Użytkownik*: fachowiec spoza grupy trygera RPC wykorzystujący pewne elementy systemu do własnych celów.



Rysunek C.1. Diagram zadań

Zadania dotyczą przeprowadzania różnego typu testów aparatury oraz jej inicjalizacji. Testy obejmują weryfikację połączeń pomiędzy układami posiadającymi BS (VIT, *Virtual Interconnect Test* [16]), testy grup układów bez BS za pomocą otaczających je układów z BS (VCCT, *Virtual Component and Cluster Test* [20,23]) oraz testy układów z BS posiadających fizyczny dostęp do pinów (BICT, *Boundary In-Circuit Test*).

Standard IEEE 1149.1 umożliwia również przeprowadzenie pewnych testów funkcjonalnych urządzeń. Takie testy można by przeprowadzać korzystając z instrukcji INTEST, BIST (*Built-In Self Test*) lub testować pewne układy bez BS (np. pamięć) poprzez podanie odpowied-

nich pobudzeń przez otaczające układy posiadające BS (VCCT). Istnieje również możliwość obserwacji urządzeń z BS podczas ich normalnej pracy (instrukcja SAMPLE). Można jej użyć do przeprowadzania pewnych prostych testów podczas przebiegu eksperymentu. Zadanie to jest jednak dość trudne do realizacji, ze względu na problem synchronizacji zegara systemowego z zegarem BS (TCK). Kolejną trudność stanowi fakt, że system trygera będzie działał w trybie przetwarzania potokowego.

Większość z wymienionych testów przeprowadzana by była okresowo, w czasie dłuższych przerw w pracy detektora. Krótkie testy kontrolujące integralność łańcucha BS oraz testy na zwarcia (podzbiór testów VIT) mogłyby być wykonywane przed każdym użyciem detektora podczas inicjalizacji.

Nie wszystkie z wymienionych na diagramie zadań są konieczne do funkcjonowania trygera.

Konieczna jest inicjalizacja układów PAC. Aby to wykonać niezbędna jest możliwość niskopoziomowego dostępu do BS w układach PAC. Jest to też element niezbędny podczas fazy projektu trygera.

Automatyczne, dokładne testy połączeń na płytkach wydają się również być koniecznym elementem ze względu na wielkość trygera RPC. W tak dużym systemie elektronicznym z pewnością będą powstawać błędy w połączeniach na płytkach oraz pomiędzy różnymi modułami. Wykonanie testów VIT jest minimalną formą realizacji tego zadania. Dobrze by było posiadać również możliwość wykonania testów VCCT oraz BICT.

Aby łatwiej sklasyfikować wymienione zadania, posłużę się modelem warstwowym testów z użyciem BS [24].

6. Model warstwowy

W celu łatwiejszego zrozumienia testów z użyciem BS oraz zaprojektowania systemu wygodnie jest podzielić typowy test na kilka oddzielnych poziomów abstrakcji (Rysunek C.2).

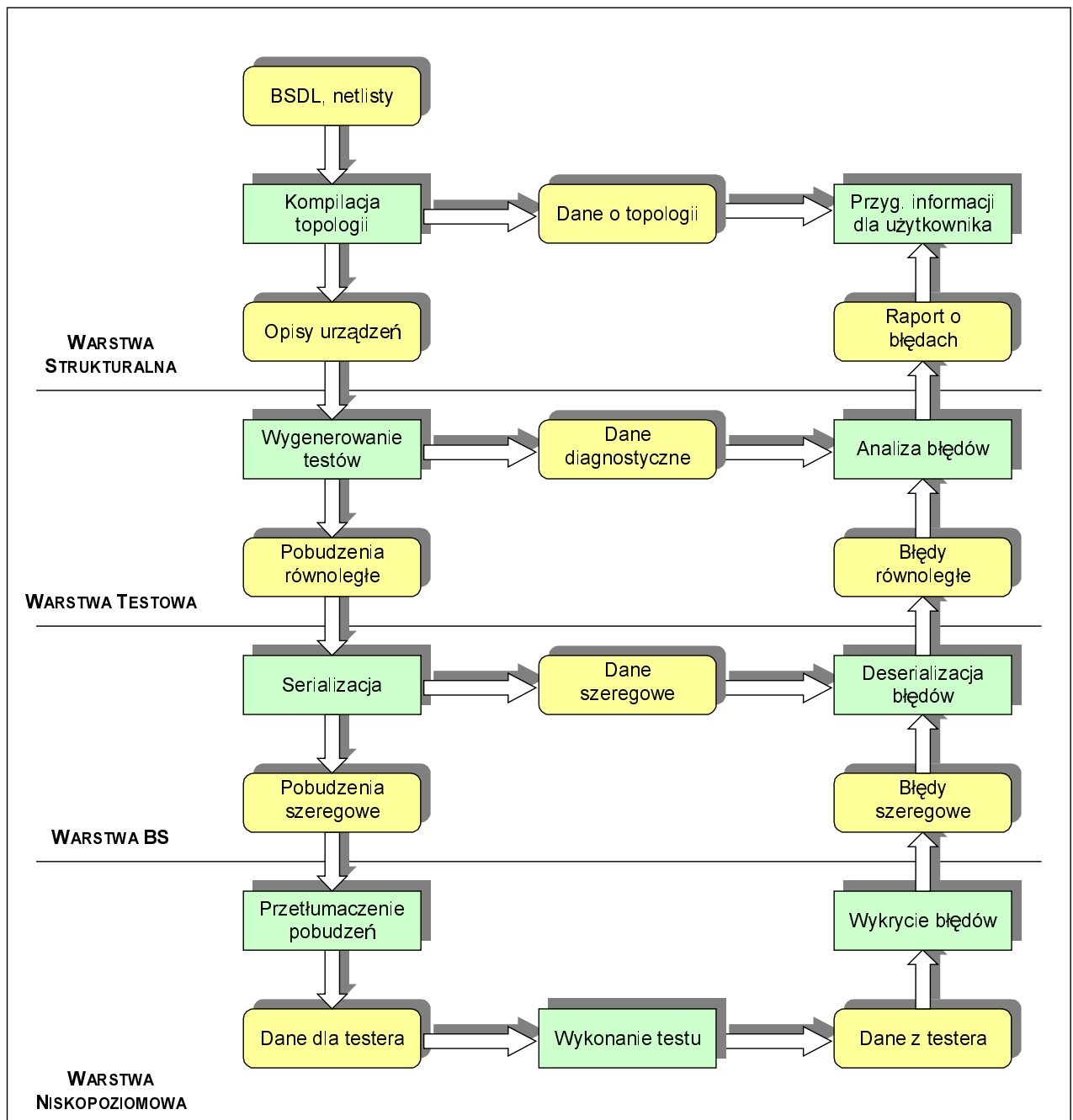
Problem testowy rozpoczyna się opisem testowanego urządzenia przy pomocy plików o takich formatach jak: BSDL*, HSDL†, plikach opisujących listy połączeń (np. EDIF) itp. Pliki te muszą być skompilowane i struktura testowanego urządzenia zapamiętana w pewien dogodny dla

* BSDL – *Boundary Scan Description Language* (IEEE 1149.1b-1994). Język wywodzący się z VHDL, służący do opisu struktury mechanizmów BS w układzie scalonym.

† HSDL – *Hierarchical Scan Description Language*. Rozszerzenie BSDL o możliwość opisu połączeń urządzeń na poziomie płyt i modułów.

danego systemu sposób.

Następnie przygotowywane są testy urządzenia. Testy te składają się z listy równoległych pobudzeń oraz z danych diagnostycznych służących do interpretacji odpowiedzi urządzenia na podane pobudzenia. Testy te mogą pochodzić z bibliotek testowych dla komponentów urządzenia, z symulacji pewnych części urządzenia lub z analizy struktury połączeń.



Rysunek C.2. Model warstwowy

Do tej pory znajomość BS nie była nam potrzebna z wyjątkiem informacji, które elementy urządzenia umożliwiają dostęp do sygnałów za pomocą BS.

W następnym etapie dane reprezentujące równoległe pobudzenia muszą zostać uszeregowane, tzn. zamienione na łańcuchy bitów, które powinny zostać wpompowane do łańcucha BS przy odpowiednich ustawieniach kontrolerów TAP.

Pobudzenia szeregowo muszą następnie być przetłumaczone na format specyficzny dla używanego testera (kontrolera BS). Gdy tester wykona test, wyniki są przetwarzane na postać szeregową, a następnie na postać równoległą.

Błędy w postaci równoległej mogą być teraz analizowane bez odniesienia do BS z wykorzystaniem danych diagnostycznych wygenerowanych podczas tworzenia testu. Odpowiedni proces tworzy następnie raport o błędach posługując się nazwami konkretnych elementów urządzenia.

Z powyższej analizy wynika, że cykl testowy można podzielić na cztery warstwy: warstwę strukturalną, warstwę testową, warstwę BS oraz warstwę niskiego poziomu. Procesy należące do tej samej warstwy pełnią analogiczne funkcje i wykorzystują do tego specyficzne dla danej warstwy dane. I tak np. procesy z warstwy strukturalnej nie potrzebują danych opisujących algorytmy testowe, czy używany do testów kontroler BS. Podobnie procesy z warstwy BS nie muszą znać nazw płyt ani nazw testowanych układów, które zawarte są w warstwie strukturalnej.

Pewne części powyższego modelu mogą być zgrupowane w jednym programie. Dla przykładu wygodnie jest całą warstwę niskiego poziomu zaimplementować w tym samym module. W profesjonalnym systemie firmy Teradyne deserializacja, analiza błędów i przygotowanie informacji dla użytkownika realizowane są przez jeden program.

Przy dokonywaniu podziału naszego systemu na moduły należy oprócz wygody i przejrzystości oprogramowania uwzględnić kwestię standardów przemysłowych. Standardy określają formaty zapisu pewnych danych. Takimi formatami są wspomniane wcześniej EDIF oraz BSDL, jak również SVF pozwalający zdefiniować szeregową postać pobudzeń testowych. Istnieją również propozycje dotyczące zapisu błędów w postaci szeregowej (DTFD [28]), oraz danych z warstwy testowej: pobudzeń równoległych, danych diagnostycznych oraz błędów równoległych. Zgodność ze standardami przemysłowymi pozwoliłaby na wykorzystanie profesjonalnych narzędzi w celu np. generowania testów płyt lub analizy błędów.

Spróbujmy teraz spojrzeć ponownie na zadania omówione w rozdziale 5. Prawie wszystkie z nich dotyczyć będą warstwy testowej tzn. zaimplementowanie ich będzie związane z roz-

wojem tej warstwy. Wyjątek stanowi tu oczywiście zadanie *dostęp niskopoziomowy*, które będzie zrealizowane w warstwie niskiego poziomu i BS. *Inicjalizacja układów PAC* dotyczyć będzie warstwy BS.

7. Obecny stan prac

Obecnie posiadamy już pewne elementy systemu. Są to: kontroler BS, biblioteki realizujące warstwę niskiego poziomu oraz warstwę BS i program do inicjalizacji układów PAC. Poniżej zamieszczono krótkie charakterystyki wybranych elementów.

a) Kontroler BS

Kontroler BS stanowi automat synchroniczny, który można traktować jako skrajnie uproszczony mikroprocesor. Został on zrealizowany w programowalnym układzie Altera FLEX i posiada własną pamięć. Od strony komputera kontroler widziany jest jako pewien obszar pamięci (o pojemności 64k słów 16 bitowych) oraz trzy rejestry.

Kontroler dysponuje następującymi rozkazami zapisywanymi w pamięci jako 16-bitowe słowa:

Lp	Nazwa rozkazu	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	DANE DO PRZESŁANIA PRZEZ ŁAŃCUCH BSDL	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2	KONIEC PROGRAMU	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
3	POLECENIE ZMIANY TRYBU KONTROLERA TAP (RRRR – KOD TRYBU KONTR. TAP)	1	0	0	1	R	R	R	R	R	R	R	R	R	R	R	R
	Przejście kontr. TAP do stanu Reset	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Przejście kontr. TAP do stanu Run-Test/Idle	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	Przejście kontr. TAP do stanu Shift-IR	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
	Przejście kontr. TAP do stanu Shift-DR	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
4	PRZESŁANIE NNN+1 BITÓW PRZEZ ŁAŃCUCH (F _R – FLAGA WYSYŁANIA DANYCH Z PAMIĘCI, F _W – FLAGA ZAPISU ODEBR. DANYCH DO PAMIĘCI)	1	1	F _R	F _W	N	N	N	N	N	N	N	N	N	N	N	N
	Przesłanie cykliczne NNNN+1 bitów w zamkniętej pętli bez zapisu odebranych bitów	1	1	0	0	N	N	N	N	N	N	N	N	N	N	N	N
	„Wpompowanie” NNNN+1 bitów ignorując odebrane bity	1	1	1	0	N	N	N	N	N	N	N	N	N	N	N	N
	„Wpompowanie” NNNN+1 bitów zapisując odebrane bity w pamięci	1	1	1	1	N	N	N	N	N	N	N	N	N	N	N	N
	Przesłanie cykliczne NNNN+1 bitów w zamkniętej pętli z zapisem odebranych bitów	1	1	0	1	N	N	N	N	N	N	N	N	N	N	N	N
5	PRZEJŚCIE KONTR. TAP ZE STANU SHIFT-IR LUB SHIFT-DR WRAZ Z WYSŁANIEM BITU N	1	0	1	0	X	X	X	X	X	X	X	X	1	F _R	F _W	N

Kontroler wyposażony jest również w trzy rejestry: rejestr statusu, rejestr adresu zapisu i

rejestr adresu odczytu, kontrolujące przebieg pracy kontrolera.

b) Oprogramowanie warstwy niskiego poziomu

Do obsługi kontrolera BS utworzono specjalną bibliotekę *libbscontr* (Rysunek C.3). Klasy w niej zdefiniowane zostały zaprojektowane w taki sposób, by późniejsze zmiany w oprogramowaniu związane ze zmianą używanego kontrolera lub sposobu komunikacji z nim (np. zamiana interfejsu VME lub sposobu adresowania płyt) zredukować do minimum. W tym celu posłużono się klasami abstrakcyjnymi (*VME*, *BSController*) pełniącymi rolę interfejsów do obiektów klas właściwych (*BSControllerWZ* – klasa obsługująca obecnie używany kontroler, *VMEBit3* – klasa zapewniająca dostęp do VME za pomocą interfejsu Bit3).

Funkcje zadeklarowane w interfejsie *BSController* zostały dobrane w taki sposób, by umożliwić łatwe wpisywanie i odczytywanie dowolnych danych wchodzących i wychodzących z łańcucha BS, oraz manipulację kontrolerami TAP, niezależnie od budowy kontrolera. Pozwalają one również na łatwą integrację z analizatorem składniowym SVF. Funkcje te dzielą się na dwie grupy:

a) Funkcje programowania kontrolera:

ProgramStart() – funkcja, którą należy wywołać przed każdą zmianą konfiguracji kontrolera (wpisaniem nowego programu)

ProgramEnd() – funkcja, którą należy wywołać po każdej zmianie konfiguracji kontrolera

ProgramExecute() – funkcja, którą należy wywołać w celu wykonania programu znajdującego się w kontrolerze.

b) Funkcje – operacje na BS – wpisywanie programu do kontrolera. Do nich należą m.in.

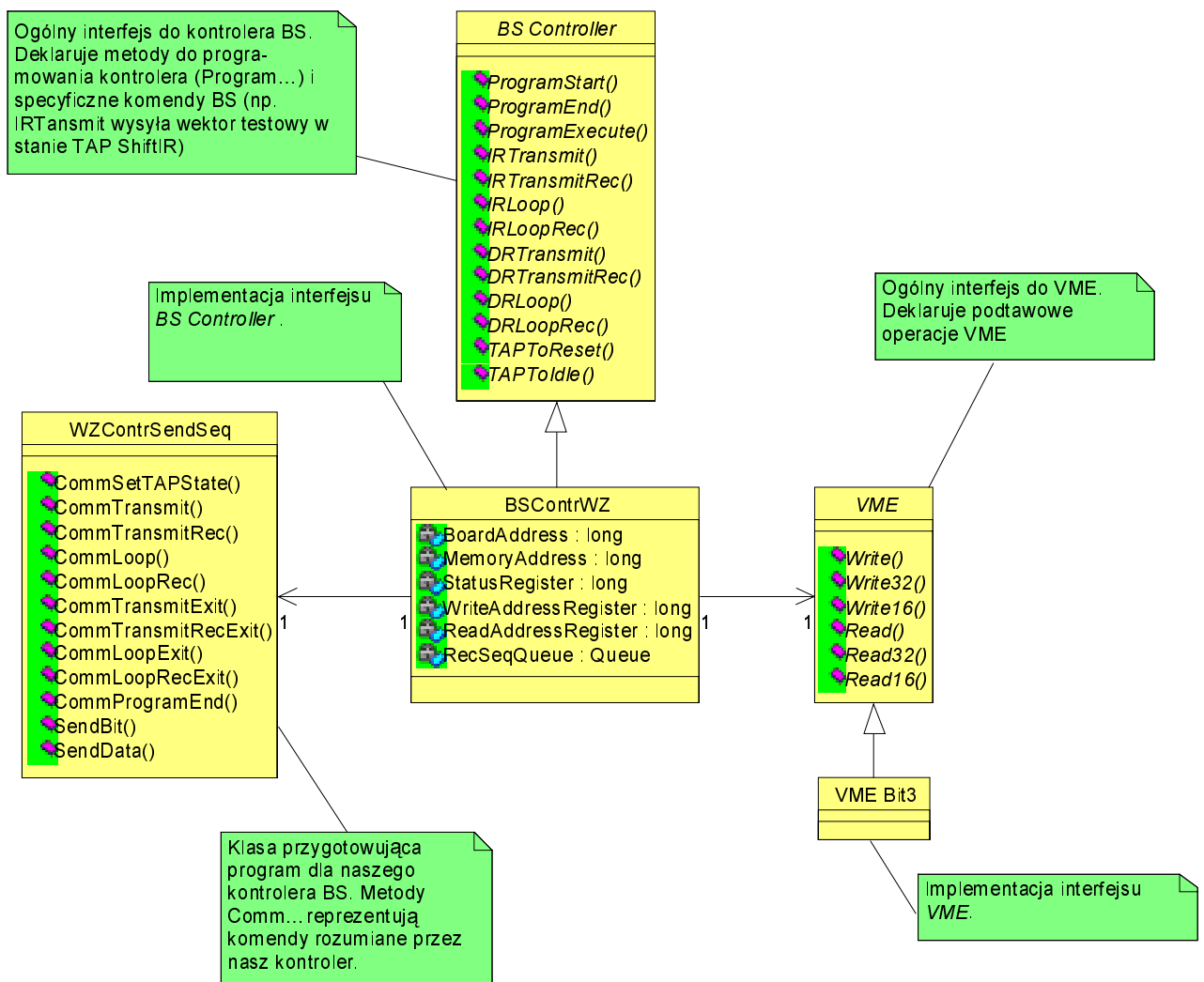
IRTransmit(), *DRTransmit()* – wysłanie danych przez Instruction/Data Register

IRTransmitRec(), *DRTransmitRec()* – wpompowanie danych do łańcucha z jednoczesnym zapisem wychodzących z łańcucha danych

IRLoop(), *DRLoop()* – wpompowanie do łańcucha danych z niego wychodzących (praca w zamkniętej pętli – połączenie TDO testowanego modułu z TDI)

IRLoopRec(), *DRLoopRec()* – wpompowanie do łańcucha danych z niego wychodzących wraz z odczytem wychodzących z łańcucha danych

TAPtoReset(), *TAPtoIdle()* – ustawienie kontrolera TAP w wybrany stan



Rysunek C.3. Struktura interfejsu do obsługi kontrolera BS (*libbscontr*)

Użycie biblioteki przedstawia następujący przykład:

```
try {
    // utworzenie obiektu-interfejsu do kontrolera umiejscowionego
    // na płycie VME o adresie #A
    TBSContr * bsc = new TBSContrWZ(0xa);

    // przygotowanie 'miejsca' na odebranie dwóch 10-bitowych wektorów z
    // łańcucha BS
    TScanVector vec1(10, ' '), vec2(10, ' ');

    //////////////////////////////////////
    // zaprogramowanie kontrolera//
    //////////////////////////////////////
    bsc->ProgramStart(); // początek programowania kontrolera
```

```

bsc->TAPtoReset();    // ustawienie kontrolera TAP do stanu RESET

bsc->IRTransmitRec( "1000101110", vec1 ); // przesłanie bitów
// "1000101110" przez rejestry instrukcji układów (TAP w
// stanie SHIFT_IR) i zapisanie odebranych bitów w vec1

bsc->DRTransmitRec( "1111000011", vec2 ); // przesłanie bitów
// "1111000011" przez rejestry danych układów (TAP w stanie
// SHIFT_DR) i zapisanie odebranych bitów w vec2

bsc->ProgramEnd();    // koniec programowania kontrolera

// uruchomienie kontrolera - wykonanie programu i
// rzeczywiste zapisanie odebranych danych do vec1 i vec2
bsc->ProgramExecute();

// wypisanie na ekranie komputera otrzymanych danych
cout << "vec1=" << vec1 << endl;
cout << "vec2=" << vec2 << endl;

}
catch( TException& e ) {
    cerr << e.what() << endl;
}

```

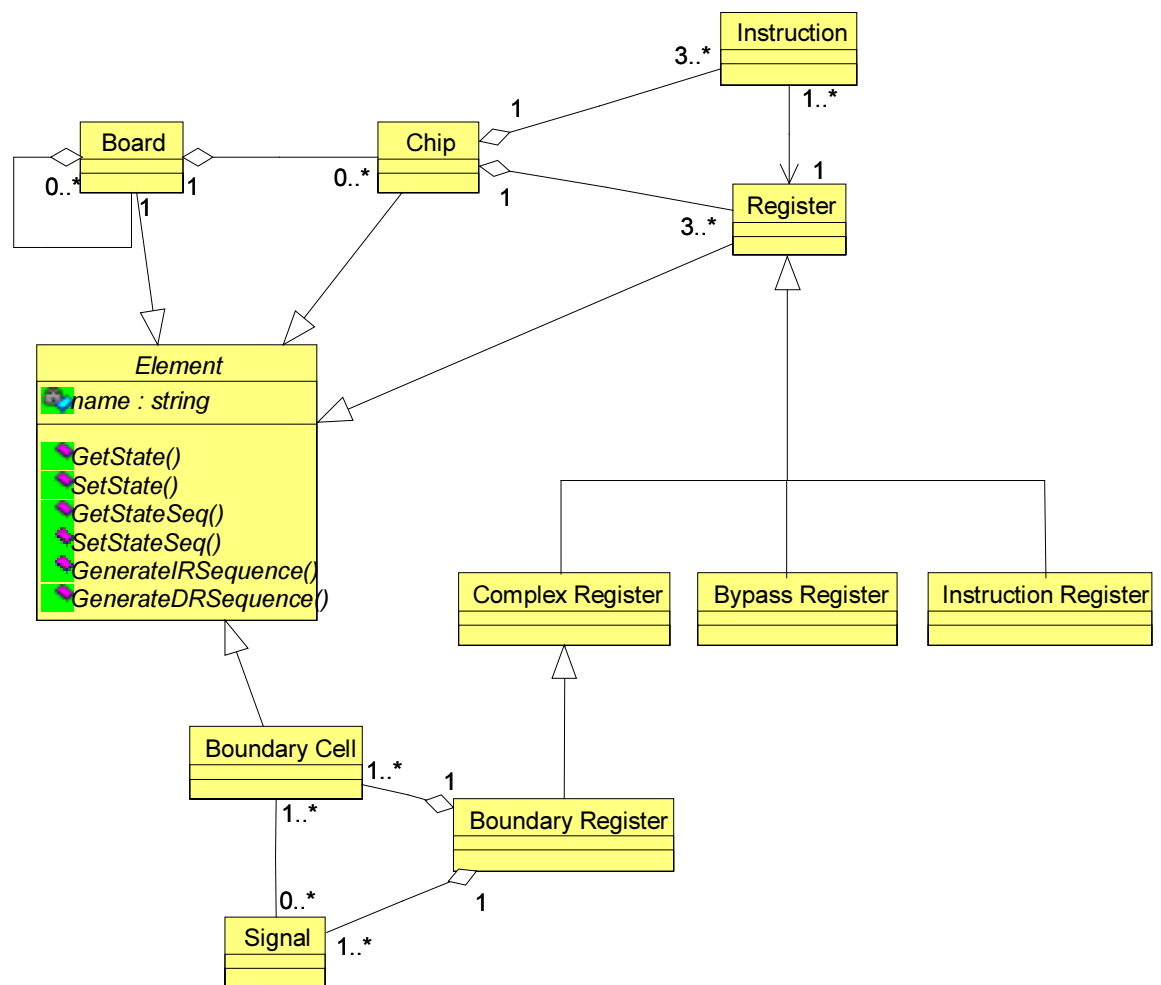
Jak widać biblioteka ta skutecznie ukrywa szczegóły związane z konstrukcją kontrolera BS, jednocześnie oferując interfejs pozwalający na bezpośrednie połączenie jej z warstwą BS.

c) *Biblioteka libbslayout*

W warszawskiej grupie CMS stworzono również bibliotekę *libbslayout*, która stanowi podstawę do implementacji warstwy BS. Pewne jej elementy mogą być również wykorzystane przez warstwę strukturalną i testową w celu reprezentacji struktury BS testowanego urządzenia.

Biblioteka pozwala odtworzyć w pamięci komputera całą strukturę mechanizmów BS urządzenia, w celu zamiany pobudzeń w postaci równoległej (czyli list stanów pinów układów, które chcemy wymusić) na formę szeregową (tj. ciągi bitów, które należy wpompować do łańcucha BS przy odpowiednich ustawieniach kontrolerów TAP) i odwrotnie: szeregową postać danych wypompowanych z łańcucha BS zamienić na równoległą postać stanów pinów układów.

Rysunek C.4 przedstawia strukturę tej biblioteki pomijając szczegóły implementacji. Główne klasy w tej bibliotece odpowiadają poszczególnym elementom. Klasa *Board* odpowiada płytkom drukowanym. Na każdej płytce mogą znajdować się kolejne płytki (w ten sposób można tworzyć moduły składające się z większej ilości płyt) lub układy scalone (*Chip*). Każdy układ scalony składa się z kolei z przynajmniej trzech rejestrów (*Instruction Register*, *Bypass Register*, *Boundary Register*). Boundary Register jest złożony z listy swoich komórek (*Boundary Cell*), listy sygnałów (*Signal*) oraz listy odpowiadających im pinów.

Rysunek C.4. Struktura biblioteki *libbslayout*

Prawie wszystkie z wymienionych dotychczas klas realizują interfejs *Element*. Funkcje *SetState...* umożliwiają ustawienie wybranego elementu w żądany stan, funkcje *GetState...* zwracają stan wybranego elementu, funkcje *Generate...Sequence* zwracają sekwencje bitów (*ScanVector*), które należy wysłać w celu ustawienia elementu w żądany (ustawiony za pomocą funkcji *SetState...*) stan, a funkcje *Interprete...Sequence* interpretują otrzymane z łańcucha BS sekwencje bitów tak, by stan wybranych elementów można było odczytać za pomocą funkcji *GetState...*.

Metody *SetState...* i *GetState...* pozwalają przeprowadzać operacje nie tylko na elemencie (klasie), w którym są zaimplementowane, ale również na elementach „na nim się znajdujących”. Pierwszym z argumentów tych funkcji jest *relatywny adres elementu* (np. „board1#chip_a#pin#A15”). W ten sposób, mając wskaźnik tylko do głównej płyty, można dotrzeć do każdego elementu.

Użycie tej biblioteki demonstruje następujący przykład:

```
try {
    //////////////////////////////////////
    //////////////////////////////////////
    //////////////////////////////////////
    // utworzenie płytki o nazwie "board" z miejscem na 2 elementy
    // znajdujące się na niej
    TBoard board("board",2);

    // utworzenie i ustawienie na płytce dwóch układów: PAC'a
    // i Altery
    board.AddElement( *new TPac("pac") );
    board.AddElement( *new TAltera("alt") );

    //////////////////////////////////////
    //////////////////////////////////////
    //////////////////////////////////////

    // "wykonanie" instrukcji PAC'a USER_7 wybierającej jako rejestr
    // danych rejestr DR2
    board.SetState("pac","USER_7");

    // wpisanie do PAC'a do multiplexera P001_MX1 bitów 1010, a do
    // demultiplexera D016_01 bitów 0101
    board.SetStateSeq("pac#DR2#P001_MX1","1010");
    board.SetStateSeq("pac#DR2#D016_01","0101");

    // wykonanie instrukcji Altery SAMPLE/PRELOAD
    board.SetState("alt","SAMPLE");

    // ustawienie dwukierunkowych pinów Altery T12 i R12 odpowiednio
    // do stanów: wyjście/wysoki, wyjście/niski.
    board.SetState("alt#pin#T12","o1");
    board.SetState("alt#pin#R12","o0");

    //////////////////////////////////////
    //////////////////////////////////////
    //////////////////////////////////////
    // WYGENEROWANIE SEKWENCJI BITÓW, KTÓRE NALEŻY WYSLAC DO LANCUCHA, //
    // BY ZOSTAŁY USTAWIONE POWYZSZE STANY (STANY PINÓW NIE OKRESLONYCH//
    // POWYZEJ BĘDĄ USTAWIONE ZGODNIE Z WARTOŚCIAMI 'SAFE' ODPOWIEDNICH//
    //                                     KOMOREK BS)                                //
    //////////////////////////////////////

    // wygenerowanie sekwencji do wysłania w stanie TAP SHIFT_IR
    TScanVector irseq;
    board.GenerateIRSequence( irseq );
    cout << "ir=" << irseq << endl;

    // wygenerowanie sekwencji do wysłania w stanie TAP SHIFT_DR
    TScanVector drseq;
    board.GenerateDRSequence( drseq );
    cout << "dr=" << drseq << endl;
}
catch( TException& e ) {
    cerr << "Exception: " << e.what() << endl;
}
```

d) *Program interbs*

Program *interbs* jest próbą realizacji zadania *dostęp niskopoziomowy*. Stanowi on prosty interfejs do bibliotek *libbscontr* oraz *libbslayout*. Pozwala on na interaktywną pracę z urządzeniem przy użyciu paru prostych poleceń lub wykonanie wcześniej przygotowanych skryptów. Program zna strukturę BS testowanego urządzenia, możliwe jest więc zarówno wykonywanie operacji podstawowych, jak wysyłanie określonych sekwencji bitów do łańcucha i manipulacje stanami kontrolerów TAP, jak również wykorzystanie właściwości biblioteki *libbslayout* w celu przygotowania i interpretacji wektorów testowych.

Program rozumie następujące komendy:

<code>reset</code>	ustawienie kontrolerów TAP układów w łańcuchu w stanie Reset
<code>idle</code>	ustawienie kontrolerów TAP układów w łańcuchu w stanie Run-Test/Idle
<code>shiftir</code>	ustawienie kontrolerów TAP układów w łańcuchu w stanie Shift-IR
<code>shiftdr</code>	ustawienie kontrolerów TAP układów w łańcuchu w stanie Shift-DR
<code>sir "1110..."</code>	wysyłanie podanej sekwencji bitów przez łańcuch w stanie TAP Shift-IR (programowanie rejestrów instrukcji układów)
<code>sdr "1110..."</code>	wysyłanie podanej sekwencji bitów przez łańcuch w stanie TAP Shift-DR
<code>ss <i>relatywny_adres_elementu symbol stanu elementu</i></code>	(set state) komenda używana w celu przygotowania sekwencji bitów do wysłania. Ustawia element określony relatywnym adresem w żądany stan, np.: - <code>ss chip1 BYPASS</code>: wpisanie do rejestru instrukcji układu chip1 kodu odpowiadającego instrukcji BYPASS - <code>ss chip1#BOUN_REG#pin#A1 o0</code>: ustawienie komórek rejestru brzegowego układu chip1 odpowiadających pinowi A1 tak, by pin przeszedł do stanu niskiego i pracował jako wyjście. Inne możliwe stany pinów: o1 – stan wysoki/wyjście, i0 – stan niski/wejście, i1 – stan wysoki/wejście. - <code>ss chip1#BOUN_REG#sig#codeout0 o1</code>: ustawienie komórek rejestru brzegowego układu chip1 odpowiadających sygnałowi codeout0, tak by osiągnąć stan wysoki/wyjście

`sss relatywny_adres_elementu "1110..."`

(set state as sequence) tak jak instrukcja powyżej z tą różnicą, że stan wybranego elementu określany jest sekwencją bitów. W ten sposób można wprowadzić np. nietypowe bity do rejestru instrukcji: `sss chip1#INST_REG "1101"`

`sir` wysłanie przygotowanych bitów do rejestrów instrukcji układów (wysłany zostanie taki wektor, aby ustawić rejestry instrukcji układów w taki stan, jaki został określony za pomocą np. instrukcji `ss chip1 BYPASS`) oraz zapamiętanie odebranych z łańcucha bitów (mogą być one zinterpretowane za pomocą instrukcji `gs` i `gss` opisanych poniżej)

`gs relatywny_adres_elementu`

(get state) interpretacja odebranych z łańcucha bitów. Podaje stan wybranego elementu w postaci symbolicznej (analogicznie jak w `ss`)

`gss relatywny_adres_elementu`

(get state as sequence) interpretacja odebranych z łańcucha bitów. Podaje stan wybranego elementu w postaci ciągu bitów (analogicznie jak w `sss`).

Poniżej zamieszczono dwa przykłady pracy z programem:

Przykład 1. W łańcuchu znajdują się cztery układy: dwa układy Altera FLEX[®] i dwa układy PAC w kolejności PAC-Altera-PAC-Altera licząc od strony TDO łańcucha. Wykonujemy instrukcję BYPASS dla wszystkich czterech układów i przez rejestry danych (w tym przypadku jednobitowe rejestry Bypass) przesyłamy sekwencję „1010101010101010”.

```
9:tb@suncms1:/usr1/user/tb/testy_paca/bsc>interbs pt_ap_pac_ac.bsc
PAC:pt
ALTERA:ap
PAC:pac
ALTERA:ac
interbs>reset
interbs>ss pt BYPASS
interbs>ss ap BYPASS
interbs>ss pac BYPASS
interbs>ss ac BYPASS
interbs>sir
sending sequence:11111111111111
received=10110001011000
interbs>sdr "1010101010101010"
received=0000101010101010
interbs>quit
10:tb@suncms1:/usr1/user/tb/testy_paca/bsc>
```

Na powyższym przykładzie widać, że aparatura działa dobrze. Otrzymana z rejestrów instrukcji sekwencja bitów jest zgodna z wytycznymi standardu. Standard określa, że ciąg bitów odczytywanych z rejestru instrukcji powinien rozpoczynać się bitami 10... . W naszym przypadku otrzymaliśmy sekwencje: 101 z PAC’a pt, 1000 z Altery ap, 101 z PAC’a pac oraz 1000 z Altery

ziomowej oraz BS. Kolejnym krokiem powinno być stworzenie pilotowej wersji systemu realizującej proste testy połączeń na płytach (VIT). Następnie należałoby stworzyć taki projekt systemu, by mógł on być uzupełniany o kolejne rodzaje testów np. VCCT, BFT i inne. Wtedy też powinna zostać podjęta decyzja dotycząca podziału systemu na moduły i wykorzystania standardów dotyczących zapisu danych testowych.

Dodatek D. Testy procesora PAC

1. Wstęp

W pierwszej połowie 1998 roku warszawska grupa CMS przeprowadziła testy prototypu procesora PAC. Procesor ten jest dedykowanym układem scalonym zaprojektowanym na potrzeby trygera, którego zadaniem jest odnalezienie mionu o najwyższym pędzie w danych pochodzących z komór RPC detektora CMS [29]. Jest on urządzeniem programowalnym (pozwala na zaprogramowanie kształtów śladów akceptowanych mionów i odpowiadających im pędów), synchronicznym, działającym w trybie przetwarzania potokowego.

2. Architektura układu

Dokładny opis układu można znaleźć w pracy [6]. Poniżej krótko zostały scharakteryzowane główne jego części.

PAC zbudowany jest z następujących modułów (Rysunek D.1):

- modułu pomiaru pędu
- modułu kaskadującego
- modułu Boundary-Scan.

Pojedynczy układ PAC potrafi analizować następujący zakres sygnałów pochodzących z czterech płaszczyzn detektorów RPC:

- 18 bitów danych pochodzących z płaszczyzny MS1 (najbliższej punktowi przecięcia wiązek)
- 4 bity danych z płaszczyzny MS2
- 14 bitów danych z płaszczyzny MS3
- 18 bitów danych z płaszczyzny MS4.

Moduł pomiaru pędu (Rysunek D.2) dokonuje analizy w następujących krokach:

1. Rozpoznanie potencjalnych śladów mionów.

Sygnały z komór wchodzą na wejścia 640 grup programowalnych multiplekserów (MUX). Każda grupa odpowiedzialna jest za zidentyfikowanie jednego, predefiniowanego śladu.

2. Dopuszczenie do dalszej analizy śladów najwyższej jakości.

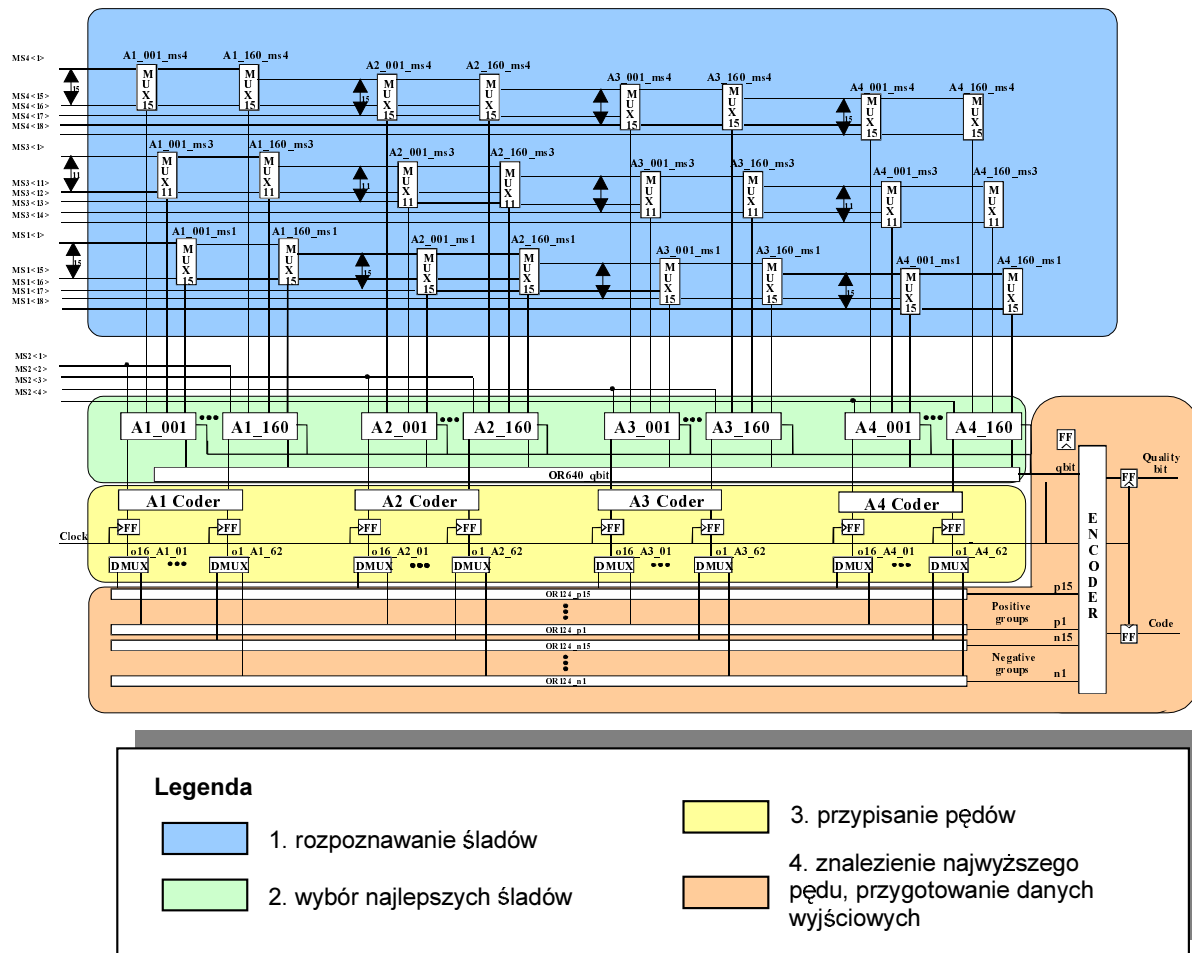
Sygnały wyjściowe z multiplekserów podlegają dalszej analizie. Jeżeli rozpoznano ślady składające się z trafień w cztery powierzchnie detektorów RPC, do dalszej analizy dopuszczane są tylko takie ślady. Ustawiany jest wtedy również wyjściowy bit świadczący o dobrej jakości danych wejściowych (Quality Bit, Qbit). W przeciwnym razie analizowane są ślady składające się z trafień w trzy powierzchnie RPC.

3. Przypisanie śladom pędu oraz znaku.

Predefiniowane ślady (grupy multiplekserów) są połączone w grupy o tym samym pędzie oraz znaku (Ai Coder). Pęd śladu (grupy) jest przypisywany przez odpowiedni demultiplekser (DMUX). Demultipleksery są programowalne: wartość przypisywanego grupie pędu definiuje się w momencie inicjalizacji układu PAC.

4. Wybór najwyższego pędu i przygotowanie danych wyjściowych.

Dane dotyczące pędów mionów analizowane są następnie przez element Encoder.. Wybierana jest najwyższa wartość pędu i zamieniana na postać binarną (cztery bity). Dopisywany jest również piąty bit (Hpt bit), informujący czy dany PAC analizuje ślady o wysokim czy niskim pędzie, bit znaku (zgodnie z ustawieniem rejestru FF1 dla śladów dodatnich i FF2 dla ujemnych) oraz bit jakości śladu (Qbit). Siedmiobitowe dane z modułu pomiaru pędu wchodzą następnie na moduł kaskadujący.



Rysunek D.2. Struktura modułu pomiaru pędu

Moduł kaskadujący pozwala połączyć maksymalnie do pięciu układów PAC w tzw. kaskadę, co umożliwia powiększenie analizowanego segmentu oraz poszerzenie zakresu pędów.

Moduł BS spełnia dwie role: funkcjonalną oraz testową. Przede wszystkim służy on do zaprogramowania układu. W tym celu wprowadzono pięć dodatkowych rejestrów danych (BST_Data_Register1, ..., BST_Data_Register5) oraz instrukcje, które je wybierają (USER_7, ..., USER_B).

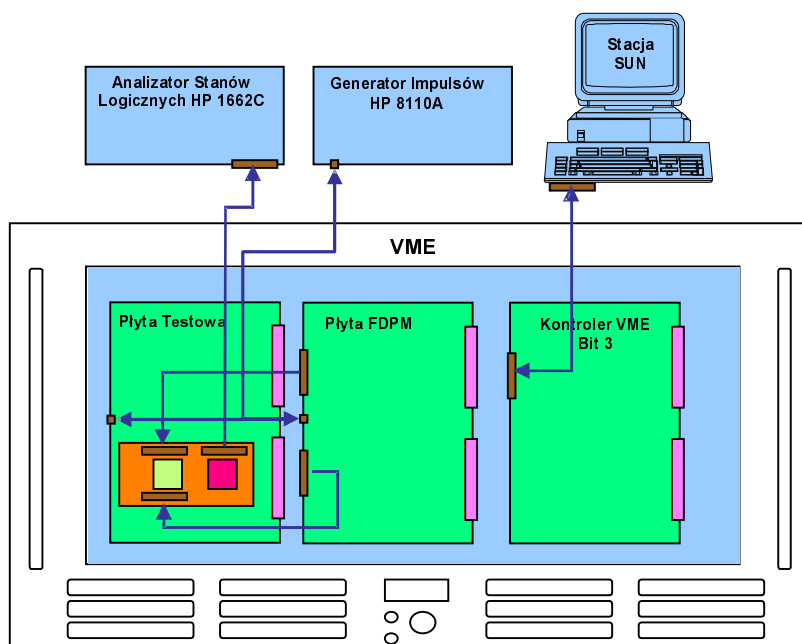
Za pomocą rejestru BST_Data_Register1 programuje się moduł kaskadujący, ustawia maski dla sygnałów wejściowych oraz wyjściowych modułu pomiaru pędu, definiuje bity znaku oraz wysokiego pędu (Hpt). Rejestry BST_Data_Register2..5 służą do programowania multiplexerów rozpoznających ślady mionów oraz demultiplexerów przypisujących grupom śladów odpowiednie pędy.

Funkcje testowe modułu BS dotyczą wykorzystania instrukcji obowiązkowych oraz in-

strukcji INTEST i IDCODE.

3. Stanowisko testowe

W celu sprawnego testowania układów PAC przygotowano specjalne stanowisko, składające się z kasy VME i kontrolera, płyty testowej, płyty FDPM, analizatora stanów logicznych, generatora impulsów oraz stacji roboczej SUN (Rysunek D.3).



Rysunek D.3. Stanowisko testowe do badania procesora PAC

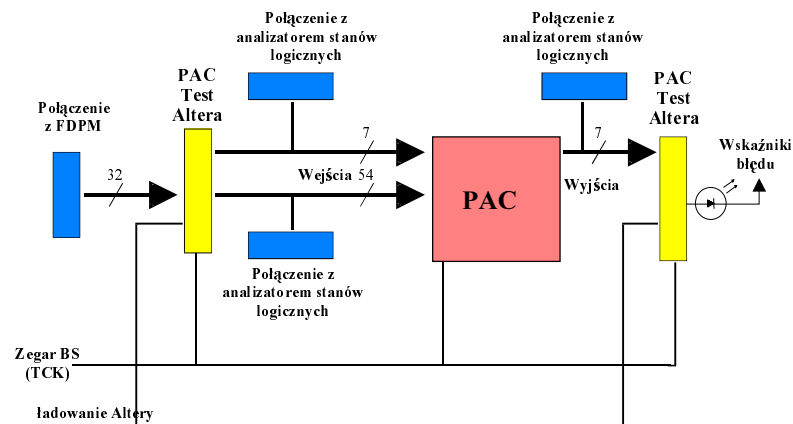
Na płycie testowej znajduje się m.in. kontroler BS, programowalny układ Altera (PAC test Altera, PT), testowany procesor PAC oraz szereg diod świecących informujących o przebiegu testów (Rysunek D.4). Altera PT przygotowuje sygnały wejściowe dla PACa oraz analizuje jego odpowiedzi. W przypadku błędu przerywa test i zapala odpowiednie diody. Sygnały testowe dla układu PAC Altera PT pobiera z generatora FDPM. W jednym takcie zegara FDPM dostarcza jedynie 32 bity danych. Z tego powodu wprowadzono dwa możliwe tryby pracy dla Altery PT:

1. Szybki: tylko 32 bity wejściowe podawane na PACa pobierane są z FDPM. Pozostałe mają stałe wartości. Odpowiedź PACa nie jest kontrolowana przez Alterę PT.
2. Pełny: dane testowe pobierane są z FDPM w dwóch taktach zegara. Zawierają one sygnały wejściowe dla PACa (54 bity) oraz oczekiwaną odpowiedź (7 bitów). Co drugi takt zegara na PACa podawane są zerowe dane wejściowe.

Wejścia oraz wyjścia procesora PAC mogą być również obserwowane za pomocą anali-

zatora stanów logicznych.

Generator impulsów jest źródłem sygnału zegara dla płyty testowej oraz płyty FDPM.



Rysunek D.4. Schemat płyty testowej.

4. Oprogramowanie do testowania

Płyty VME są za pośrednictwem interfejsu Bit3 sterowane przez stację roboczą Sun. Zadaniem oprogramowania chodzącego na stacji jest odpowiednia konfiguracja interfejsu Bit3, ładowanie układów Altera (tzn. kontrolera BS oraz układu Altera PT), zaprogramowanie układu PAC, sterowanie płytą FDPM itp. Do tego celu przygotowano specjalne środowisko składające się z następujących programów i bibliotek (Rysunek D.5):

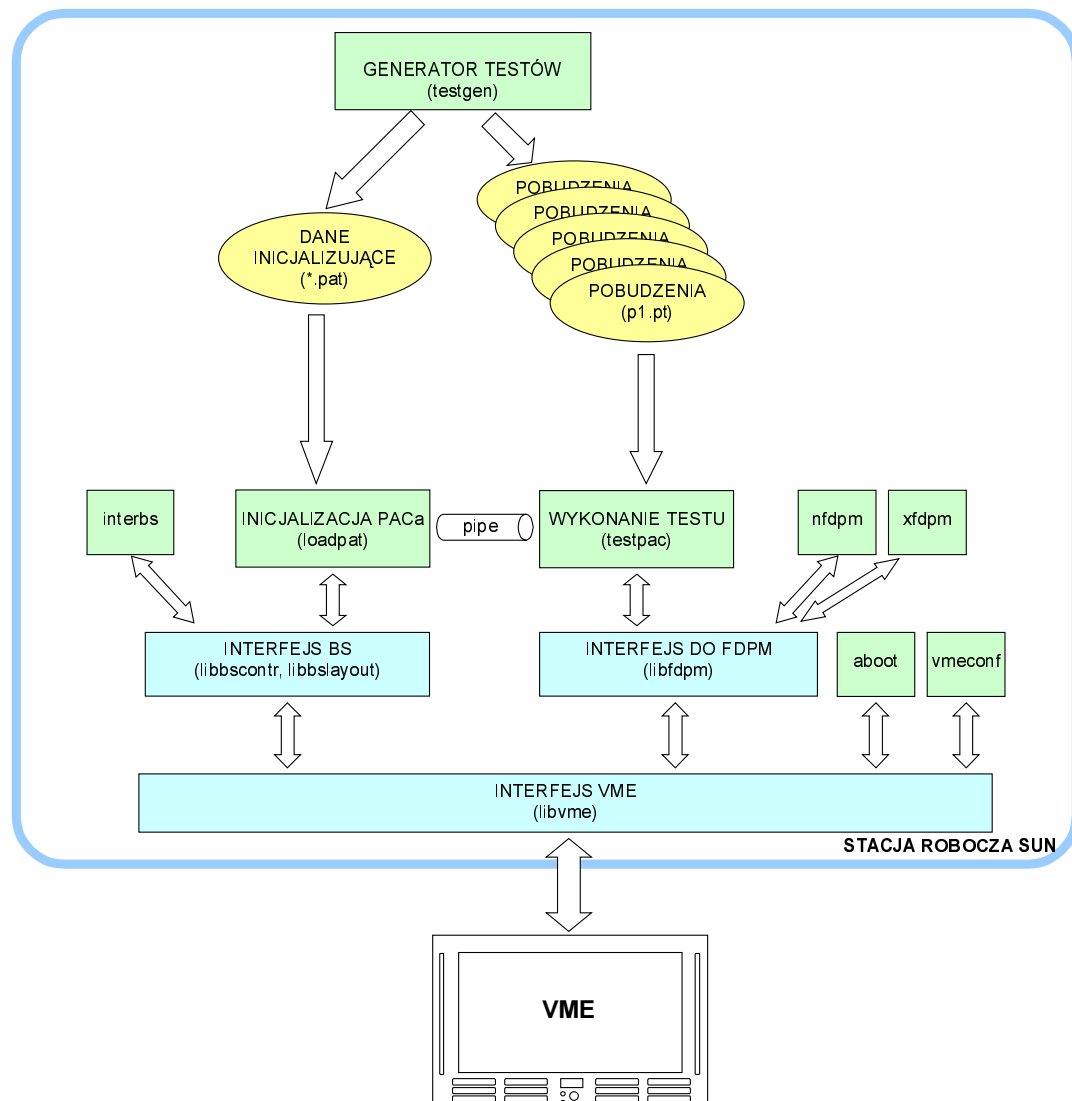
- | | |
|----------------|--|
| <i>vmeconf</i> | konfiguracja kasety VME |
| <i>aboot</i> | załadowanie układów Altera |
| <i>nfdpm</i> | tekstowy interfejs do płyty FDPM generujący sygnały na podstawie skryptów fsc (patrz Dodatek B) |
| <i>xfdpm</i> | graficzny interfejs do FDPM pozwalający na edycję skryptów fsc, ich interpretację, generowanie sygnałów, pracę w pętli (generowanie sygnałów co zadany przedział czasu) itp. (patrz Dodatek B) |
| <i>interbs</i> | interaktywne sterowanie urządzeniami BS (IEEE 1149.1) na płycie (patrz Dodatek C): |
- sterowanie kontrolerami TAP,
 - dostęp do wszystkich rejestrów,
 - interpretacja łańcucha (odczyt i ustawianie stanów poszczególnych komó-

	rek łańcucha, lub odpowiadających im pinów układów),
	- możliwość wykonywania dowolnych instrukcji,
	- symboliczna reprezentacja instrukcji, stanów pinów itd.,
	- praca z wieloma układami w łańcuchu na raz
<i>testgen</i>	generowanie testów: skryptów .pt z wymuszeniami oraz danych inicjujących dla procesora PAC (.pat)
<i>loadpat</i>	zaprogramowanie procesora PAC poprzez łańcuch BS plikami .pat
<i>testpac</i>	wykonywanie testów procesora PAC na podstawie skryptów .pt
<i>libvme</i>	klasy C++ umożliwiające dostęp do VME
<i>libfdpm</i>	zestaw klas C++ pozwalający na łatwe programowanie fdpm w celu generowania sygnałów, ukrywająca szczegóły związane ze strukturą rejestrów i pamięci przed użytkownikiem (patrz Dodatek B)
<i>libbscontr</i>	zestaw klas C++ stanowiący interfejs do kontrolera BS, m.in. deklaruje abstrakcyjną klasę TBSContr, oraz definiuje dziedziczącą z niej klasę TBSContrWZ, stanowiącą interfejs do używanego przez naszą grupę kontrolera (patrz Dodatek C)
<i>libbslayout</i>	zestaw klas C++ takich klas jak TChip, TRegister, TPin, TBoundaryCell, itp. pozwalających na utworzenie struktury łańcucha BS w pamięci komputera, służących do przygotowywania i interpretacji łańcuchów BS (patrz Dodatek C).

W pierwszej fazie testów, kiedy należało „nauczyć się” procesora, używano głównie programów uniwersalnych, tzn. aplikacji używanych również do innych, niż niniejsze testy celów. Okazało się, że dobrze pełniły one swoją rolę podczas testowania poszczególnych obszarów działania procesora PAC: program *interbs* okazał się pomocny przy wstępnym testowaniu działania mechanizmów BS, natomiast program *xfdpm*, dzięki możliwości łatwej modyfikacji generowanych sygnałów, umożliwił poznanie niuansów związanych z dynamiką testowanego układu.

W drugiej fazie testów, której zamierzeniem było dokładne przetestowanie wszystkich bloków układów PAC, testy były całkowicie zautomatyzowane. Za strategię i przygotowanie testów odpowiedzialny jest program *testgen*. Generuje on dwa rodzaje plików: pliki *.pat* zawierające dane inicjalizacyjne dla procesora PAC (m.in. definiujące ślady mionów i ich pędy), oraz odpowiadające im grupy plików *.pt* zawierające dane o wymuszeniach i oczekiwanych kodach wyników. Pliki *.pt* są następnie interpretowane przez program *testpac*. Jeśli konieczna jest ponowna inicjalizacja układu PAC (informacja o rodzaju predefiniowanych śladów zawarta jest na początku każdego pliku *.pt*) *testpac* używa do tego celu połączonego z nim programu *loadpat*. Następnie

wykorzystując uniwersalną bibliotekę *libfdpm* generuje odpowiednie sygnały za pomocą płyty FDPM.



Rysunek D.5. Schemat oprogramowania do testowania procesora PAC

5. Strategia testów

Metody testowania układów scalonych są rozwijane przez przemysł od wielu lat. Powstało wiele teorii opisujących systemowe podejście do tego problemu [11], jak również istnieją narzędzia, które w sposób automatyczny generują testy układów (tzw. ATPG – Automatic Test Pattern Generation). Jednakże ze względu na brak dostępu do takich narzędzi przez warszawską grupę CMS, nietypową budowę procesora PAC oraz jego olbrzymi rozmiar, zdecydowano się na wykonanie testów samodzielnie.

Rzeczą niemożliwą jest przetestowanie tego typu układu poprzez podawanie na wejścia wszystkich możliwych kombinacji pobudzeń i obserwowanie reakcji procesora – jest zbyt wiele takich kombinacji. Konieczne jest wybranie takich pobudzeń, które będą testować konkretne elementy układu. Zrezygnowano z testowania pojedynczych bramek i połączeń między nimi, ze względu na ich bardzo dużą liczbę. Ograniczono się na sprawdzeniu poprawnego wykonywania podstawowych funkcji przez grupy bramek tworzących główne elementy zaznaczone na rysunku D.2.

Za pomocą specjalnie do tego celu napisanego programu *testgen* przygotowano kilka serii testów, z których każda testuje inną grupę elementów układu. Procesor testowano „od tyłu”: najpierw przeprowadzano testy elementów odpowiedzialnych za końcowe etapy analizy, a dopiero potem za wcześniejsze. Takie podejście pozwala m.in. sprowadzić do minimum liczbę różnych rodzajów inicjalizacji układu.

1. Nazwa testu: Encoder

Inicjalizacja układu: Zaprogramowanie części demultiplekserów w ten sposób, by mógł zostać wygenerowany każdy kod pędowy (1..15) (zarówno dodatni jak i ujemny). Multipleksery zaprogramowane są w taki sposób, by każdy ślad odpowiadał dwóm grupom multiplekserów (i dwóm demultiplekserom): jednej odpowiadającej cząstkom dodatnim, a drugiej ujemnym.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom.

Co jest testowane:

- logika układu za Encoderem
- generowanie wszystkich kodów przez Encoder
- wybieranie wyższego kodu dla par różniących się o 1
- wybieranie dodatniego kodu dla par o tym samym pędzie

2. Nazwa testu: Pos/Neg

Inicjalizacja układu: Dwa rodzaje inicjalizacji: w jednym każdy ślad interpretowany jest jako ślad dodatni (rejstry FF1 i FF2 ustawione na 0), w drugim wszystkie ślady są ujemne

(FF1=1 i FF2=1). W obydwu rodzajach wszystkie multipleksery i demultipleksery są zaprogramowane.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom.

Co jest testowane:

- Poprawność działania rejestrów FF1 i FF2 oraz logiki za tymi rejestrami przenoszącej informację o znaku mionu.

3. Nazwa testu: Hpt

Inicjalizacja układu: Rejestr Hpt ustawiony na 1 (we wszystkich innych testach jest on wyzerowany). Wszystkie multipleksery i demultipleksery są zaprogramowane.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom.

Co jest testowane:

- Poprawność działania rejestru Hpt (FF3) oraz logiki za tym rejestrem.

4. Nazwa testu: Dmux

Inicjalizacja układu: 16 rodzajów inicjalizacji tak, by sprawdzić generowanie każdego kodu przez każdy demultiplekser. Multipleksery zaprogramowane w taki sposób, by było możliwe wybranie każdego demultipleksera.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom.

Co jest testowane:

- Działanie elementów OR124 (patrz Rysunek D.2) dla wszystkich wejść w stanie FALSE lub dokładnie jednego w stanie TRUE.
- Generowanie wszystkich kodów przez każdy z demultipleksarów.

5. Nazwa testu: Qbit

Inicjalizacja układu: Dwa rodzaje inicjalizacji.

W pierwszym wszystkie multipleksery odpowiadające dodatnim śladom mionów zaprogramowane są w taki sposób, by rozpoznawały miony na podstawie uderzeń w cztery lub trzy powierzchnie detektorów (wszystkie multipleksery w grupie odpowiadającej jednemu śladowi zaprogramowane są na niezerowe wartości). Każda grupa multiplekserów ma swojego odpowiednika wśród multiplekserów definiujących ślady ujemnych cząstek, rozpoznającego ten sam ślad na podstawie uderzeń w tylko trzy powierzchnie RPC (jeden multiplekser w grupie ustawiony na zero).

Demultipleksery dla dodatnich śladów generują kod 1, natomiast dla ujemnych kod 2.

W drugim typie inicjalizacji multipleksery i demultipleksery odpowiadające śladom dodatnich cząstek zamienione są rolami z multiplekserami i demultiplekserami dla ujemnych cząstek.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom podając uderzenia w cztery powierzchnie RPC. (PAC powinien więc za każdym razem odpowiedzieć kodem 1 z ustawionym Qbitem.)

Co jest testowane:

- Generowanie sygnału dobrej jakości (Qbit) przez każdy z układów Ai_nnn
- Działanie elementu OR640_qbit (patrz Rysunek D.2) dla dokładnie jednego z wejść ustawionego na TRUE
- Wybieranie śladów lepszej jakości przez elementy Ai_nnn
- Działanie logiki za elementem OR640_qbit przenoszącej informację o jakości śladu

6. Nazwa testu: Mux

Inicjalizacja układu: 15 rodzajów inicjalizacji. Przy każdej inicjalizacji wszystkie grupy multiplekserów są zaprogramowane i każda z nich definiuje inny ślad. Każda następna inicjalizacja programuje każdy z multiplekserów, aby stawał się aktywny przy pobudzeniu kolejnego wejścia. Wszystkie demultipleksery są zaprogramowane. W ten sposób może zostać przetestowane sprawne działanie multiplekserów dla rozpoznawania wszystkich wejść.

Pobudzenia: Podanie po kolei sygnałów odpowiadających wszystkim zaprogramowanym śladom.

Co jest testowane:

- Poprawność działania wszystkich multiplekserów dla rozpoznawania wszystkich wejść przy dokładnie jednym wejściu aktywnym.

7. Nazwa testu: Mux/Zero

Inicjalizacja układu: Trzy rodzaje inicjalizacji. Każda grupa multiplekserów jest tylko częściowo zaprogramowana, nie definiując śladu złożonego z uderzeń nawet tylko w trzy powierzchnie RPC. Wszystkie demultipleksery są zaprogramowane.

Pobudzenia: Podanie takich pobudzeń by został wygenerowany niezerowy kod pędowy w przypadku zwarcia pomiędzy sygnałami z różnych powierzchni RPC

Co jest testowane:

- istnienie zepsutych multiplekserów o zawsze aktywnych wyjściach, lub aktywnych wyjściach przy jakimkolwiek wejściu aktywnym
- zwarcia pomiędzy sygnałami z różnych powierzchni RPC.

Poniższa tabela przedstawia główne parametry testów:

Lp.	Nazwa	Liczba inicjalizacji	Liczba wektorów testowych
1	Encoder	1	56
2	Pos/Neg	2	2560
3	Hpt	1	1280
4	Dmux	16	7936
5	Qbit	2	1280
6	Mux	15	38400
7	Mux/Zero	3	128

Przetestowanie jednego układu zajmowało około jednej minuty. Większość czasu pochłaniała interpretacja plików *.pat* oraz *.pt*. Format tych plików został tak dobrany, by był łatwo rozumiany przez człowieka i był niezależny od testera. Konwersja tych plików na format specyficzny dla używanego testera lub chociażby SVF dla plików *.pat*, powinna skrócić ten czas do paru sekund.

6. Wyniki testów i podsumowanie

Przetestowanych zostało dziesięć procesorów PAC. Znalaziono pewne błędy w projekcie układu (powtarzające się w każdym układzie) oraz parę wad produkcyjnych w pojedynczych układach.

Błędy w projekcie układu:

- sygnały ze stacji MS2 są niepotrzebnie negowane,
- błędy w działaniu logiki dynamicznej układów: działanie procesorów silnie zależy od częstości oraz wypełnienia zegara; gdy aktywny jest Qbit, dane dotyczące pędu mionu rozlewane są na parę taktów zegara; pobudzenia podawane na wejścia muszą być przedzielane w czasie pobudzeniami zerowymi (w jednym takcie zegara podawane są dane, w następnym same zera, potem znowu dane itd.).

W dwóch układach znalaziono wady produkcyjne. W jednym z nich rejestr Boundary-Scan BST_Data_Register5 był popsuty, w drugim ustawienie sygnału MS4<14> nie powodowało oczekiwanej reakcji odpowiedniego multipleksera.

Opisana w tym dodatku strategia testów oraz przygotowane podczas nich wzorce testowe (zbiory plików inicjalizujących oraz pobudzeń i oczekiwanych odpowiedzi procesora PAC), chociaż okazały się efektywne przy badaniu tego prototypu procesora, powinny być rozwijane. Będą one wykorzystywane przez producenta układów w celu wyeliminowania wadliwych egzemplarzy. Powyższe testy z małym prawdopodobieństwem wykrywają choćby zwarcia na wejściach multiplekserów, nie testują one również Encodera ani elementów OR (OR640_qbit, OR124...) dla więcej niż dwóch wejść aktywnych. Nie został również w pełni przetestowany moduł kaskadujący ani moduł BS.

Podziękowania

Chciałbym serdecznie podziękować wszystkim, którzy przyczynili się do powstania niniejszej pracy. W szczególności Panu profesorowi dr hab. Janowi Królikowskiemu za skierowanie mnie na tak interesujący temat i wspomaganie mnie swoimi cennymi wskazówkami.

Jestem wielce zobowiązany panu Maciejowi Kudle za udostępnienie mi wielu rysunków zamieszczonych w tej pracy oraz za szereg wartościowych uwag.

Dziękuję również całej warszawskiej grupie CMS, a w szczególności panom Krzysztofowi Poźniakowi, Maciejowi Górskiemu, Andrzejowi Fenglerowi i Krzysztofowi Kierzkowskiemu za współpracę na polu badań opisanych w tej pracy.

Panu Wojciechowi Zabołotnemu dziękuję za nieocenioną pomoc we wstępnej fazie prac nad oprogramowaniem Boundary-Scan.

Bibliografia

1. M.Gorski, G.Iaselli, M.Konecki, J.Krolkowski, I.M.Kudla, A.Liuzzi, M.Pietrusinski, K.Pozniak, W.Zabolotny, A.Ranieri, G.Wrochna, *Test Bench for CMS RPC Muon Trigger System*, sierpień 1996, niepublikowane
2. I.M.Kudla, *Status of PAC ASIC project and Test Bench for Muon (RPC) Trigger System for CMS*, TRIDAS Meeting, 12 March 1996
3. Michał Pietrusiński, *Stanowisko testowe do badanie trygera mionowego RPC dla eksperymentu CMS przy LHC*, niepublikowane
4. Maciej Gorski, Ignacy M. Kudla, Krzysztof T. Pozniak, *Resistive Plate Chamber (RPC) based Muon Trigger System for CMS experiment - Data Compression/Decompression System (Poster Session)*
5. M.Gorski, M.Konecki, J.Krolkowski, I.M.Kudla, M.Lewandowski, A.Ranieri, G.Wrochna, *Trigger Electronics for Muon RPC Chambers in CMS*
6. Zbigniew Jaworski, Ignacy M. Kudla, Wiesław Kuzmich, Mariusz Niewczas, *Resistive Plate Chamber (RPC) based Muon Trigger System for the CMS experiment - PAttern Comparator ASIC*, Nuclear Instr. and Meth. In Phys. Res. Sect. A (419)2-3 (1998) pp. 707-710
7. A. Fucci et al., *Fast Dual Port Memory*, CERN/ECP 93-4 (Jan 1993)
8. G. Antchev, D. Gigi, *MAG – FDPM User's Manual version 1.0*, RD 12, CERN July 29, 1993
9. S. Centro et al., *M.A.G. Memory Address Generator*, CERN
10. M. Demoulin, *LabFDPM. Fast Dual Port Memory LabVIEW library*, RSTB technical note, CERN/ECP (29 April 1992)
11. Neil H. E. Weste, Kamran Eshraghian, *Principles of CMOS VLSI design: a systems perspective*. Addison-Wesley Publishing Company 1994, ISBN 0-201-53376-6
12. Marek Adamus, Krzysztof Poźniak, *Propozycja rozwoju systemu elektronicznego detektora Veto Wall w oparciu o model Elementarnego Bloku Funkcjonalnego*
13. *Impact of JTAG/1149.1 Testability on Reliability*, Texas Instruments, SCTA041A, January 1997
14. *IEEE Std 1149.1 (JTAG) Testability Primer*, Texas Instruments 1997, Semiconductor Group
15. *IEEE Standard Test Access Port and Boundary Scan Architecture*, IEEE Std 1149.1-1990

16. Kenneth P. Parker, *The Boundary-Scan handbook*, Kluwer Academic Publishers 1992, ISBN 0-7923-9270-1
17. W. C. Bruce, M. G. Gallup, G. Giles and T. Munns, *Implementing 1149.1 on CMOS Microprocessors*, Proceedings, International Test Conference, Nashville TN, pp 879-886, Oct 1991
18. A. Halliday, G. Young and A. Crouch, *Prototype Testing Simplified by Scannable Buffers and Latches*, Proceedings, International Test Conference, Washington DC, pp 174-181, Aug 1989
19. P. Hansen, *The Impact of Boundary-Scan on Board Test Strategies*, Proceedings, ATE&I Conference, Boston MA, pp 35-40, June 1989
20. R. Adams, *Automatic Test Generation, Diagnostics, and Design Tools for Virtual Component and Cluster Testing*, NEPCON West'93 Proceedings of the Technical Program, Feb 1993
21. M. F. Lefebvre, *Functional Test and Diagnosis: A Proposed JTAG Sample Mode Scan Tester*, Proceedings, International Test Conference, Washington DC, pp 294-303, Sept 1990
22. P. Hansen, *Hierarchical Test Generation: Taking Boundary Scan to the Next Level*, Proceedings of IEEE Autotestcon, 1996
23. G. Wedge, *Using Boundary Scan With a Fault Dictionary To Test and Diagnose Clusters of Non-Scan Logic*, Proceedings of IEEE Autotestcon, 1996
24. G. Wedge, Tom Conner, *A Roadmap for Boundary-Scan Test Reuse*, IEEE International Test Conference, 1996
25. Wiesław Kuźmich et al., *VLSI implementation of the RPC Pattern Comparator (PAC) ASIC for the CMS experiment – feasibility study*, CMS TN/96-003, 11 Jan 1996
26. Martin Fowler with Kendall Scott, *UML Distilled: Applying the Standard Object Modeling Language*, The Addison-Wesley Longman 1997, 0-201-32563-2
27. Grady Booch, James Rumbaugh, and Ivar Jacobson, *Unified Modeling Language Users Guide*, The Addison-Wesley Longman 1997, 0-201-571168-4
28. Teradyne, Inc., *Digital Test Failure Language*, VICTORY 2.30 Reference Manual, 1996
29. CMS Collaboration, *The Compact Muon Solenoid Technical Proposal*, CERN/LHCC 94-38, 15 December 1994
30. G.L. Bencze et al., *Nucl. Instrum. Methods A* 340 (1994) 466

Index

A

aboot · 31, 66

Ai Coder · 63

B

BFT · 60

BICT · 47, 48

bootall · 31, 32

Boundary-Scan · *Zobacz* IEEE 1149.1

BSDL · 17, 48, 50, 51

BYPASS · 13, 15, 57, 58, 59

C

Capture-DR · 15, 16

D

daq · 31

DEFM · 39, 40, 41

DEFSEQ · 38, 40, 41, 42

DEFSIG · 38, 39, 41

detektor

CMS · 1–2, 4, 10, 20, 46, 54, 61, 68, 75,
76

VETO WALL · 1, 7, 8

ZEUS · 1, 2–3, 7, 19

DMUX · 63

DTFD · 50

E

EDIF · 48, 50

ENDM · 39, 40, 41

ENDR · 41, 42

ENDSEQ · 38, 40, 41, 42

ENDSIG · 38, 39, 41

evscan · 31

EXTEST · 13, 15, 16

F

faza docelowa · 4

faza projektowania · 1, 4, 6, 36, 46

fdpm · 31, 37, 67

FDPM · 7, 10–11, 22, 25, 31, 37, 38, 39, 40,
41, 42, 44, 65, 66, 68, 75

FEC · 4, 20

fsc · 37, 38, 40, 42, 43, 44, 66

G

gs · 58, 59

gss · 58

H

HIGHZ · 13, 16

Hit-Readout · 7, 8

Hpt · 63, 64, 70

HSDL · 48

I

IDCODE · 13, 16, 65

idle · 57

IEEE 1149.1 · 7, 11–18, 46, 47, 48, 66, 75,
76

interbs · 57, 58, 59, 66, 67

INTEST · 13, 16, 47, 65

J

JTAG · *Zobacz* IEEE 1149.1

K

kontrola

nadażna danych · 7

quasi-bieżącą stanu aparatury · 7

quasi-nadażną danych · 7

trygera podczas trwania eksperymentu · 7

kontroler BS · 51

kontroler TAP · 13, 14, 51, 53, 54

L

LDEMUX · 5, 20

libbscontr · 52, 53, 57, 67

libbslayout · 54, 55, 57, 67

libfdpm · 11, 67, 68

LMUX · 5, 20

loadpat · 67

M

mfdpm · 31

MUX · 63

N

nfdpm · 11, 37, 42, 66

P

PAC · 4, 20, 23, 24, 25, 26, 27, 28, 29, 31,
46, 48, 51, 56, 58, 59, 61–64, 65, 66, 67,
68, 71, 73, 75, 76, 82

Q

Qbit · 63, 70, 71, 73

Quality Bit · 63

R

RC · 30, 32, 34

rc-daq · 31

rc-evscan · 31

rc-monitor · 31

rc-pac · 31

rc-syn · 31

rc-tbset · 31, 32

rc-trg · 31

readout · 31

rejestr

brzegowy · 13, 15

bypass · 13, 15

danych · 13, 15, 54, 58

identyfikacyjny · 13

instrukcji · 13
 REPEAT · 41, 42
 reset · 57, 58, 59
 RPC · 2, 4, 20, 23, 25, 46, 61, 63, 71, 72, 75, 76
 run controller · 46
 RUNBIST · 13, 16
run-controller · 30, 31, 32, 34, 35

S

SAMPLE/PRELOAD · 13, 15, 48, 56, 59
 sdr · 57, 58, 59
 sekcja
 definicji sekwencji · 38
 układu sekwencji · 38, 39
 SENDSEQ · 38, 39, 40, 41
 SET · 32, 34, 38, 39, 40, 41
 shiftdr · 57
 Shift-DR · 15, 51, 57
 shiftir · 57
 Shift-IR · 15, 51, 57
 sir · 57, 58, 59
 ss · 57, 58, 59
 sss · 57, 58
 stanowisko testowe
 trygera RPC · 4, 21
 trygera RPC · 21–24
 trygera VETO WALL · 7, 8
 STEPS · 38, 39, 40, 41
 SVF · 50, 52
syn · 31
syn-test · 31
syn-testx · 31
 system wyzwalania · 1, 6, 10, 17

T

TAP · 13, 14, 15, 16, 50, 51, 52, 54, 56, 57, 66
 TCK · 13, 14, 15, 48
 TDI · 13, 14, 15, 16, 52
 TDO · 13, 14, 15, 16, 52, 58
testgen · 67, 69
testpac · 67
 testy
 ad hoc · 12, 17, 18
 docelowych elementów w laboratorium · 6
 modułów · 12
 plyt · 12
 podczas trwania eksperymentu · 7
 prototypów · 6, 20
 trygera przed użyciem detektora · 7
 układów scalonych · 12
 własne · 7, 8, 12, 16
 TMS · 13, 14, 15
trg · 31
 TRST · 13
 tryger · 1, 6, 20, 34
 mionowy · 2
 mionowy RPC · 1, 2, 4, 5, 20–21, 35, 46, 48, 75

V

VCCT · 47, 48, 60
 VIT · 47, 48, 60
vmecconf · 31, 66

W

testowa · 50

warstwa

BS · 50, 51

niskiego poziomu · 50, 51

strukturalna · 50, 54

X

$\propto dpm$ · 11, 37, 42, 43, 45, 66, 67

