Açık Kaynak Kodlu SysML Modelleme Yazılımı

Selçuk İyikalender

YÜKSEK LİSANS TEZİ

Elektrik-Elektronik Müh. Anabilim Dalı

Mayıs 2010

An Open Source SysML Modelling Software

Selcuk Iyikalender

MASTER OF SCIENCE THESIS

Department of Electrics and Electronics Engineering

May 2010

Açık Kaynak Kodlu SysML Modelleme Yazılımı

Selçuk İyikalender

Eskişehir Osmangazi Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik-Elektronik Müh. Anabilim Dalı
Elektronik Bilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır

Danışman: Yrd. Doç. Dr. Kemal Özkan

Mayıs 2010

ONAY

Elektrik-Elektronik Müh. Anabilim Dalı Yüksek Lisans öğrencisi Selçuk İyikalender'in YÜKSEK LİSANS tezi olarak hazırladığı "Açık Kaynak Kodlu SysML Modelleme Yazılımı" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Danışman **:** Yrd. Doç. Dr. Kemal ÖZKAN

İkinci Danışman **: -**

Yüksek Lisans Tez Savunma Jürisi:

Üye : Yrd. Doç. Dr. Kemal Özkan

Üye : Prof. Dr. İdris DAĞ

Üye : Yrd. Doç. Dr. Nihat ADAR

Üye : Yrd. Doç. Dr. Erol SEKE

Üye : Yrd. Doç. Dr. Selçuk CANBEK

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. Nimetullah BURNAK

Enstitü Müdürü

ÖZET

Bu tez çalışmasında, UML 2 temel alınarak sistem mühendisliği uygulamaları için genişletilen bir diyagram dili olan SysML ile modellemenin yapılabileceği açık kaynak kodlu yazılım geliştirilmiştir. Geliştirilen bu yazılım ile, özellikle sayısal elektronik sistemlerinin yapısal modelinin oluşturulması ve oluşturulan bu yapısal model üzerinden de VHDL tasarımının otomatik üretilmesi amaçlanmıştır. Örnek uygulamalar üzerinde yapılan çalışmalarda, yapısal modelden otomatik olarak VHDL tasarımı üretilebileceği görülmüştür. Bununla birlikte mevcut SysML modelleme yazılımları da bu tez çalışması kapsamında incelenmiştir.

Anahtar Kelimeler: SysML, Model Tabanlı Geliştirme, Yazılım, UML, UML 2, Sistem, Tasarım, Kod Üretimi, VHDL, MOF, DIM

SUMMARY

In this thesis, an open source software that allows modeling with SysML, a diagram language developed by extending UML-2 for systems engineering applications, is developed. This software specifically aims structural modeling of digital electronic systems and automatic generation of VHDL designs over these models. Through experimental designs, VHDL designs are generated automatically from structural models. Existing SysML modeling software are also comparatively examined.

Keywords: SysML, Model Driven Development, Software, UML, UML 2, System, Design, Automated Code Generation, VHDL, MOF, DIM

TEŞEKKÜR

Yüksek Lisans tez çalışmalarımda, bana danışmanlık ederek, beni yönlendiren ve her türlü olanağı sağlayan danışmanım Yrd. Doç. Dr Kemal ÖZKAN' a teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

ŞEKİLLER

Sayfa

ÇİZELGELER

Sayfa

KISALTMALAR

Kısaltma	Açıklama
UML	Unified Modeling Language
SysML	Systems Modeling Language
VHDL	VHSIC Hardware Definition Language
OMG	Object Management Group
GTK	GIMP Toolkit
GIMP	GNU Image Manipulation Program
INCOSE	International Council on Systems Engineering
SE RFP	Systems Engineering Request for Proposal
SE DSIG	Systems Engineering Domain Special Interest Group
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
SoC	System on Chip
GNU	GNU's Not Unix

1 GİRİŞ

1.1 SysML Hakkında

Günümüzde modelleme dilleri, araçları ve teknikleri sistem mühendisliği uygulamalarında geniş yelpazede kullanılmaktadır. UML (Unified Modelling Language) 'in, yazılım endüstrisinde kullanılan modelleme dillerini birleştirdiği gibi, SysML(Systems Modeling Language, Sistem Modelleme Dili) 'de sistem mühendisliğinde ayrı olarak kullanılan modelleme dillerini birleştirmeyi amaçlamaktadır. SysML kısaca sistem mühendisliği uygulamaları için geliştirilmiş karmaşık sistemlerin spesifikasyon, analiz, tasarım, doğrulama ve geçerlemesini kapsayan genel amaçlı bir modelleme dilidir. Söz konusu sistemler donanım, yazılım, bilgi, süreçler, personel ve tesisleri içerebilir (OMG, 2008).

SysML' in kökeni, International Council on Systems Engineering (INCOSE) Model Driven System Design (Model Tabanlı Sistem Tasarımı) çalışma grubunun tarafından UML (Unified Modelling Language) ‘in sistem mühendisliği uygulamaları için özelleştirme çalışmalarına kadar uzanır. Bu da INCOSE ve UML 'i oluşturan Object Management Group (OMG) 'in ortaklaşa olarak Temmuz 2001'de OMG Systems Engineering Domain Special Interest Group (SE DSIG) 'i kurmaları ile sonuçlanır. SE DSIG INCOSE ve ISO AP 233 çalışma gruplarının desteği ile Mart 2003 modelleme dilinin gereksinimlerini geliştirir. Bu çalışma OMG' in UML for Systems Engineering Request for Proposal (Sistem Mühendisliği için UML için Teklife Çağrı, UML for SE RFP) isimli bir dokümanla yayınlanmıştır (OMG, 2008).

Temel olarak SysML, UML 2' in özelliklerinden bir alt kümeyi kullanır ve UML for SE RFP 'de tanımlanmış olan gereksinimleri karşılayacak şekilde UML 2' in önerdiği genişletme mekanizmalarını kullanır. SysML temel olarak UML 2' i kullandığından, iki modelleme dilinin ortaklaşa kullanımı da mümkündür. Bu özellikle yazılım içeren sistemlerde bütünleşik bir modelleme ortamını sağlar (OMG, 2008).

1.2 SysML'in Avantajları

SysML 'in avantajları genel hatları ile aşağıdaki şekilde sıralanabilir:

- SysML açık bir standarttır. SysML ile ilgili birçok dokümana ücretsiz olarak rahatlıkla ulaşılabilir. Bunun yanında kullanımı için de bir bedel ödenmemektedir. Bu açıdan SIMULINK gibi ticari olan modelleme dillerine göre avantajlıdır.
- Sistem Mühendisliğinde ürün yaşam döngüsünün belirli safhalarında ihtiyaç duyulan yöntemler (gereksinimler, tasarımlar, analizler, doğrulama ve geçerleme) tek bir yapıda toplanması SysML getirdiği başlıca avantajlardır. Örneğin gereksinimler, ürün yaşam döngüsü içinde önemli bir yere sahiptir. Projelerin büyüklüğüne, kritiklik derecelerine göre gereksinimlerin belli bir sistematik içinde ele alınması önem arz etmektedir. Müşteriden gelen gereksinimlerin geliştirilen ürün içinde nerede ve nasıl karşılandığı izlenebilir olması da gereklidir. Değişen gereksinimler ve tasarım öğeleri arasında bu izlenebilirliğin korunması da gereklidir. Gereksinimler ve tasarımlar için endüstride genelde farklı yazılımların kullanılması nedeniyle (özellikle yazılımlar birbirini desteklemiyor ise) izlenebilirliği tanımlamak genelde sorunlu olabilmektedir. SysML 'in gereksinimleri de içine alan tümleşik bir ortam sunması izlenebilirlik konusunda önemli bir avantaj getirmektedir. Ayrıca gereksinim diyagramları, gereksinimlerin sistematik olarak ele alınması için de diyagram genişletmelerini de sunmaktadır.
- Büyük sistemlerin modellenmesinde, özellikle sistem farklı mühendislik uygulamalarını kapsıyorsa, çalışma grupları arasındaki veri alış verişini kolaylaştırır.
- SysML 'in temel aldığı UML profilleri de ileriye dönük veya kullanıcı ihtiyaçlarına göre özelleştirme yapabilmesine olanak sağlar.

1.3 SysML’in Dezavantajları

Temel olarak SysML 'in dezavantajları temelde UML dilinden gelmektedir ve aşağıdaki şekilde sıralanabilir:

- Örneğin UML profilleri dilin genişlemesi için kullanılsa da, bu aslında oldukça sınırlıdır. Yeni modelleme kavramlarının ve sembollerinin tanımlanması mümkün değildir. Örneğin bir elektronik devre şemasındaki transistör sembolü tanımlanamaz.
- SysML 'in etkin kullanımı, modelleme kavramları konusunda belli bir aşinalık gerektirir.
- SysML' in etkin olarak kullanımı büyük ölçüde kullanılan yazılım aracına bağlıdır.

1.4 SysML’in Elektronik Mühendisliği Alanında Kullanımı

Elektronik Mühendisliği açısından SysML’in bir çok uygulama alanı olabilir. Bu uygulama alanları genel hatları ile aşağıda sıralanmıştır:

1. ASIC, FPGA ve SoC tasarımlarının modellenmesi ve geliştirilen modellerden SystemC, VHDL, Verilog tasarımlarının üretilmesi,
2. Parametrik diyagramlar ile elektronik sistemlerin matematiksel modellerinin oluşturulması,
3. Devre kart seviyesi tasarımlarda; örneğin birden fazla karttan oluşan sistemler ve/veya kartlar üzerinde bulunan alt bileşenler için modellerin oluşturulması olarak sıralanabilir.

2 GELİŞTİRİLEN YÖNTEM

Bu tez çalışması kapsamında, ASIC, FPGA ve SoC tasarımlarının yapısal modellenmesi ve geliştirilen yapısal modellerden, VHDL tasarımının üretilmesi için bir çözüm üretilmeye çalışılmıştır. VHDL tasarımının üretimi için, SysML blok tanımlama diyagramları ve iç blok tanımlama diyagramları temel alınmıştır.

2.1 Yazılımının Özellikleri

Yazılımın temel özellikleri aşağıda sıralanmıştır:

1. C++ dilinde geliştirilmiştir.
2. Yazılım grafik kullanıcı arabirimi tabanlıdır. SysML modeli çizilerek geliştirilir.
3. Modelin ve diyagramların bilgileri tek bir XML tabanlı dosyada saklanmaktadır.
4. SysML modelinden VHDL kodu üretilmektedir.

2.2 Yazılımın Kurulması

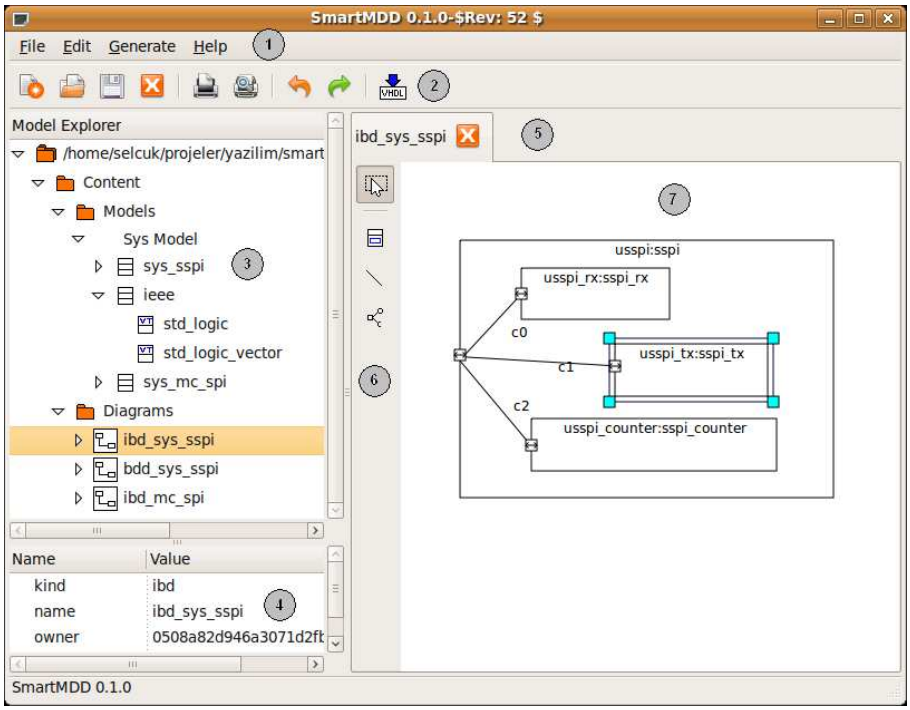
Yazılımın kurulması için kaynak kodundan inşa edilmesi gerekmektedir. İnşa işlemi için “cmake” (2.6 ve üzeri) , “make” (GNU sürümü) , “gcc” yazılımları kullanılmıştır. “cmake” yazılımı için hazırlanmış yapılandırma dosyasının (CMakeFiles.txt) içeriğinde yapılan tanımlamalar nedeni ile, inşa işleminin UNIX tabanlı işletim sistemlerinde yapılması gerekmektedir.

Yazılımın inşası için bazı dış kütüphanelerin işletim sisteminde kurulmuş olması gerekmektedir. Bu kütüphaneler aşağıda belirtilmiştir:

- Gtk+2.0 (2.20 ve üzeri sürümler kullanılmalıdır)
- Boost (1.42 ve üzeri sürümler kullanılmalıdır)
- Expat (2.0.1 ve üzeri sürümler kullanılmalıdır)

2.3 Yazılımının Kullanıcı Ara Birimi

Yazılımın grafik kullanıcı arabirimi tabanlı yapılmış olmasının sebebi SysML modelleme diyagramlarının kolayca oluşturulmasını ve değiştirilmesini sağlamaktır. Yazılımın GNOME masaüstündeki görüntüsü Şekil 2-1 ‘ de gösterilmiştir:



Şekil 2-1 Yazılımın Kullanıcı Arabirimi

Şekil üzerinde daire içinde numaralandırılmış olan kısımlar, yazılımı oluşturan görsel alt bileşenleri gösterir. Bu görsel alt bileşenlerin türleri ve işlevleri aşağıda açıklanmıştır:

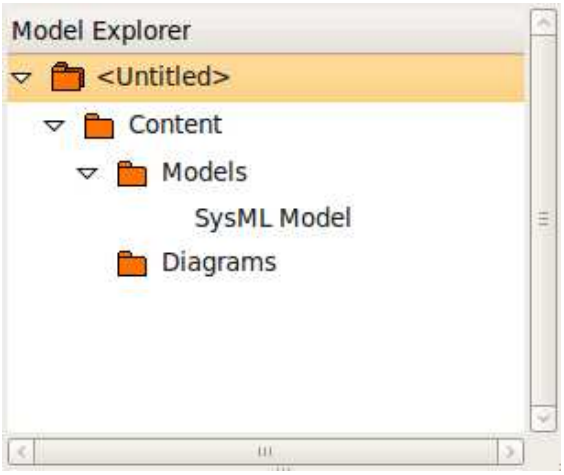
1. Menü : Yazılımın temel işlevlerini barındıran bileşendir.
2. Araç Çubuğu : Menü altında bulunan ve sık kullanılan işlevlerin, kullanıcı tarafından kolay ulaşılmasını sağlayan bileşendir.
3. Model Gezgini : SysML modelinin hiyerarşik olarak oluşturulduğu ve gösterildiği bileşendir. SysML model elemanlarının tamamı bu bileşen kullanılarak tanımlanabilir.
4. Özellik Gezgini : Model Gezgini üzerinde seçilen model elemanlarının

bilgilerinin detaylı olarak gösterildiği bileşendir.

5. Diyagram Seçim Kutucuğu : Özellikle birden fazla diyagram açılmış ise, görüntülenmesi istenen diyagramın seçilmesini sağlayan bileşendir.
6. Model Elemanı Seçim Araç Çubuğu : Diyagram üzerinde yerleştirilecek model elemanlarının bulunduğu araç çubuğudur. Diyagramın türüne göre farklı model elemanları listelenir.
7. Çizim Alanı : Kullanıcının diyagramı çizdiği bileşendir. SysML model elemanları ve model elemanları arası ilişkiler bu bileşen üzerinde tanımlanabilir.

2.4 Yazılım ile Modelin Oluşturulması

Yazılım ilk açıldığında, kullanıcı var olan bir projeyi açabilir veya yeni bir proje oluşturabilir. Kullanıcı yeni bir proje oluşturmak istediğinde, bu eylemi Menü bileşeni üzerinden “File->New” seçilerek veya Araç Çubuğu üzerinden  düğmesine basarak gerçekleştirir. Bu aşamada, Model Gezgini bileşeninde proje için bir çatı oluşturulur (Şekil 2-2).

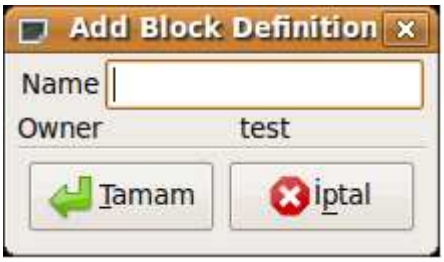


Şekil 2-2 Yeni Oluşturulmuş Model

Yeni proje oluşturulduğunda, Model Gezgini bileşeninde hiyerarşik ilişkilendirlmiş klasörlerin en üstünde proje ismi vardır. Onun altında “Content” klasörü

mevcuttur. “Content” klasörü altında “Model” ve “Diagram” adında iki alt klasör bulunur. “Model” klasörü altında bulunan “SysML Model” klasörü ise, SysML modelin oluşturulduğu klasördür. Kullanıcı bu “SysML Model” klasörünün üzerine fare imleci ile gelip ters (sağ elini kullananlar için genelde farenin sağ tuşu) tıkladığında bir açılır pencere çıkar. Kullanıcı bu açılır pencereden ihtiyaç duyduğu model elemanlarını seçerek, sistem modelini oluşturmaya başlar.

Diyagram ise, modelde bulunan bir SysML Blok elemanı üzerine ters tıklandığında gösterilen açılır pencereden seçilir. Kullanıcı, açılır pencereden diyagramı seçtiğinde, bir diyalog penceresi görüntülenir ve diyagram için gerekli olan bilgileri tanımlar (Şekil 2-3). Her diyagram bir SysML Blok elemanına aittir. Oluşturulan, diyagram eklendiğinde Model Gezgini bileşeni altındaki “Diagram” klasöründe görülebilir. Kullanıcı diyagram üzerine çift tıkladığında, diyagram Çizim Alanı bileşeninde görüntülenir.



Şekil 2-3 Blok Tanımlama Diyagramı Ekleme Penceresi

Kullanıcı, Çizim Alanı bileşeninde, yeni model elemanları ekleyebilir, var olanları görüntüleyebilir (Bu işlem, Model Gezgini üzerinde bulunan model elemanları Çizim Alanı üzerine sürükleyerek gerçekleştirilir.) ve model elemanları arasındaki ilişkileri tanımlayabilir.

Model tamamlandıktan sonra VHDL tasarımının üretimi için Araç Çubuğu bileşeninden 

düğmesine basılır.

2.5 Modelden VHDL Kod Üretimi

Tez kapsamında geliştirilen yazılım ile, VHDL tasarımı, sistemin yapısal modelini tanımlayan SysML blok tanımlama ve iç blok diyagramları temel alınarak

üretilir. Diyagramlarda tanımlanmış olan elemanların, anlamsal olarak VHDL tasarımı içinde karşılıkları vardır. Kullanılan diyagram elemanları ve bunların VHDL koduna nasıl çevrildiği ile ilgili detaylı bilgiler aşağıda verilmiştir:

2.5.1 Blok (SysML:Block) Elemanı

Blok elemanının VHDL dili karşılığı “entity” tanımlamasıdır. Örnek olarak salt blok tanımlaması ile aşağıdaki tasarım üretilir:

```
entity <blok ismi>
end <blok ismi>;

architecture behavioural of <blok ismi>
begin
end behavioural;
```

Burada <blok ismi> bir meta tanımlama olup gerçekte blok elemanının ismini içerir (SysML:Block “name” özniteliği).

2.5.2 Akış Kapısı (SysML:FlowPort) Elemanı

Akış Kapısı, VHDL varlığı (“entity”) içindeki kapı (“port”) tanımlamalarını oluşturmak için kullanılır. Örnek:

```
entity <blok ismi>
port
(
-- <akış kapısı ismi> signals
<sinyal ismi> : <yön> <sinyal tipi>
);
end <blok ismi>;
architecture behavioural of <blok ismi>
begin
end behavioural;
```

Akış kapıları temel olarak, atomik olan ve atomik olmayan olarak ikiye ayrılabilir. Atomik olan akış kapısı tek yönlüdür ve basit bir tipi olmalıdır. “SysML:Block” veya “SysML:ValueType” tanımlamaları basit tipler olarak ele alınır. Bunun nedeni bu tiplerin herhangi yön bilgisi içermeleridir. Bu nedenle VHDL tasarımındaki “<yön>” tanımlamalarını üretmek için atomik akış kapısının “direction” özniteliği kullanılır.

Atomik olan akış kapısının tipi, bir değer tipi ise (SysML:ValueType) “<sinyal ismi>” tanımlaması, “<Akış kapısının ismi>” ile “<Değer tipinin ismi>” tanımlamalarının, “_” karakteri ile birleştirilmesi ile oluşturulur.

Atomik olan akış kapısının tipi, bir blok ise (SysML:Block), “<sinyal ismi>” tanımlaması, blok altında tanımlı olan her nitelik (SysML:Property) için ayrı olarak, “<Akış kapısnın ismi>_<Blok ismi>_<Nitelik İsmi>” şeklinde oluşturulur.

Atomik olmayan akış kapıları için tip bir akış özelliği olmalıdır. Bu konu ile ilgili detay bir sonraki bölümde açıklanmıştır.

2.5.3 Akış Özelliği (SysML:FlowSpecification) Elemanı

Akış Özelliği, atomik olmayan akış kapılarının sinyalleri tanımlamak amacı ile kullanılır. Akış özelliği içinde akış tanımlamaları bulunur. Her akış tanımlaması, akışın ismi, yönü ve tipi ile ilgili bilgiler içerir. VHDL tasarımındaki kapı sinyalleri bu bilgilere göre üretilir.

2.5.4 Parça (SysML:Parts) Elemanı

Parça elemanı, ait olduğu blok elemanına karşılık olarak üretilen VHDL varlığı (“entity”) içindeki, bileşen (“component”) ve olgu (“instance”) tanımlamalarına karşılık gelir. Örnek:

```
entity <blok_ismi>
end <blok_ismi>;
architecture behavioural of <blok_ismi>
      -- Otomatik parça
```

```
        component <parçaya ait blok'un ismi>
        port
        (
        <sinyal ismi> : <yön> <sinyal tipi>;
        );
        end component;
    begin
        <parça ismi> : <parçaya ait blok'un ismi>
        port map
        (
        -- Sinyal eşlemeleri
        );
    end behavioural;
```

Her parça elemanının bir tipi ("type" özniteliği ile tanımlanır.) vardır. Bu tip bir blok (SysML:Block) elemanıdır. VHDL tasarımındaki bileşen ("component") tanımlaması parçanın tipi olan blok ve onun içinde tanımlanmış olan akış kapısı tanımlamalarından oluşturulur. Yapılan işlem bir anlamda VHDL "entity" üretmek için yapılan ile aynıdır.

<parça ismi> meta tanımlaması gerçekte akışın ismini içerir (SysML:FlowPort "name" özniteliği).

Bunun yanında parça elemanı için bir olgu ("instance") üretilir. Üretilen olgu, parçanın somut olarak gerçeklenmiş halidir.

2.5.5 İki Yönlü Bağlantı (SysML:BidirectionalConnector) Elemanı

İki yönlü bağlantı elemanları, VHDL tasarımındaki birden fazla olguyu ("instance") birbirine bağlamak amacı ile gerekli olan sinyal tanımlamalarını ve sinyal eşlemelerini oluşturmak amacı ile kullanılır. Örnek:

```
    architecture behavioural of <blok_ismi>
        -- Bağlantı için kullanılan sinyal tanımlamaları
```

```
        signal <bağlantı sinyal ismi> : <sinyal tipi>;
        -- ... diğer sinyaller
    begin
        <1.parça ismi> : <1.parçaya ait blok'un ismi>
        port map
        (
        -- Gerekli sinyal eşlemeleri
        <kapı sinyal ismi> => <bağlantı sinyal ismi>
        );

        <2. parça ismi> : <2.parçaya ait blok'un ismi>
        port map
        (
        -- Gerekli sinyal eşlemeleri
        <kapı sinyal ismi> => <bağlantı sinyal ismi>
        );
    end behavioural;
```

Kodun üretilmesi için bağlantı akış kapıları üzerinden kurulmalı ve akış kapılarının tipleri aynı olmalıdır. Gerekli olan sinyal ve eşleme tanımları akış kapılarının tipleri temel alınarak oluşturulur.

2.6 Uygulama Örnekleri ve Karşılaştırmalar

2.6.1 Örnek 1 - "Ve" Kapısı

İlk örnek mantıksal "Ve" işleminin VHDL "entity" gerçeklemesidir. Sistemin SysML diyagramı Şekil 2-4' de verilmiştir:

Şekil 2-4 “Ve” Kapısı (SysML Gösterimi)

SysML diyagramında mantıksal “Ve” işlemi, “and” isimli bir bloktan oluşturulmuş bir parça (“part”) olarak gösterilmiştir. Parçanın sol tarafında bulunan iki adet akış kapısı kapı girişleri temsil eder, sağ tarafındaki akış kapısı ise çıkışı temsil eder. Yukarıda SysML diyagramı olarak gösterilen sistemin elektronik devre sembolü, Şekil 2-5’ de verilmiştir:

Şekil 2-5 “Ve” Kapısı (Elektronik Devre Sembolü)

Görüldüğü üzere, iki gösterim arasında karmaşıklık açısından önemli bir fark yoktur. SysML diyagramından üretilmiş olan VHDL tasarımı için karşılaştırma ise Çizelge 2-1’ de verilmiştir:

Çizelge 2-1 “Ve” Kapısı için Üretilen ve Elle Yazılan VHDL Satırları (*)

Dosya	**Üretilen VHDL Satır Sayısı**	**Elle Yazılan VHDL Satır Sayısı**
and.vhd	11	1

(*) Sayıma boşluk ve yorum satırları dahil edilmemiştir.

Üretilen VHDL tasarımı, “Ve” kapısının yapısal tasarımının karşılığıdır. Elle yazılan kısmı ise “Ve” kapısının davranışını tanımlar. Aşağıda üretilen VHDL tasarımı

verilmiştir:

```
entity _and
port
(
A : in std_logic;
B : in std_logic;
C : out std_logic
);
end _and;


architecture behavioural of _and
begin
  C <= A and B;
end behavioural;
```
[1]

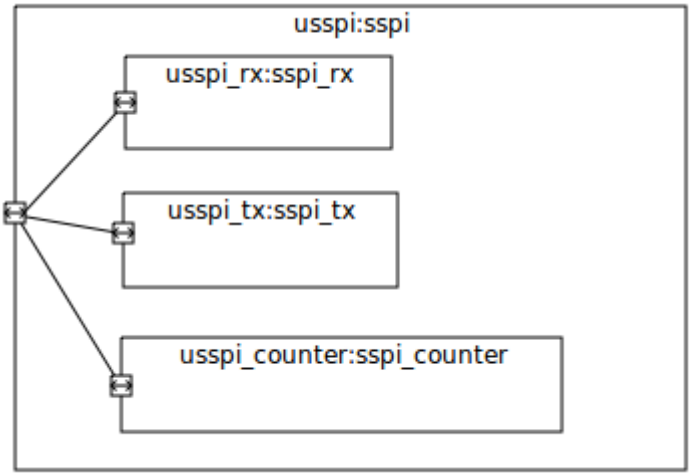
Verilen örnekte modelden VHDL kodunun üretimi sonucunda iyi bir performans elde edilmiş gibi gözükse de, gerçekte “Ve” kapısını VHDL varlığı (“entity”) olarak gerçeklemek uygulamada pek kullanışlı bir yöntem değildir. Çünkü VHDL içinde tanımlı olan “and” sözdizimi (keyword) işleminin kullanımı çoğu zaman yeterlidir.

2.6.2 Örnek 2 - Seri Arabirim Arayüzü (SPI)

İkinci örnek olarak da bir önceki örneğe göre olarak daha karmaşık bir yapıya sahip Uydu (Slave) SPI tasarımı gerçekleştirilmiştir. Bu örnek, SysML modelinin ve modelden otomatik kod üretiminin yararının daha açık olarak gösterir. Örnekte kullanılan sistem bir ana blok (“usspi”) altından tanımlanmış üç alt bloktan oluşur. Bu

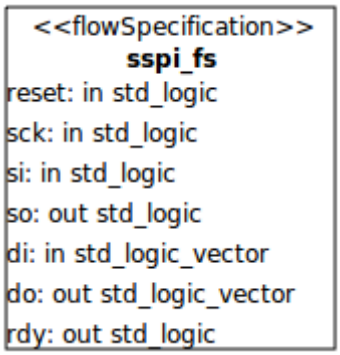
[1] VHDL tasarımında düz olarak yazılan satırlar otomatik olarak üretilmiş kısımları eğik satırlar ile elle ilave edilmiş kısımları gösterir

alt blokların üzerindeki kapılardan ana blok üzerindeki akış kapısına bağlantı kurulmuştur. Sistemin SysML diagramı Şekil 2-6’ te verilmiştir:



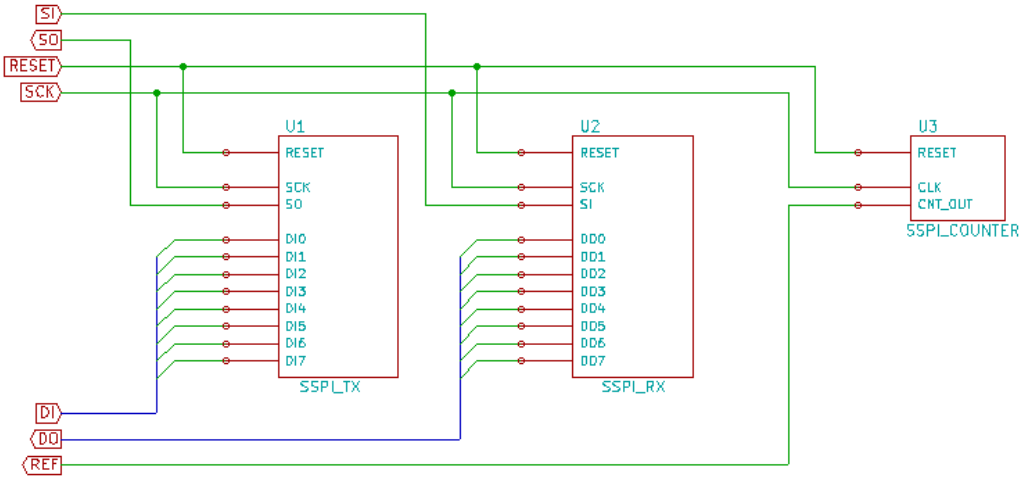
Şekil 2-6 SPI SysML İç Blok Diyagramı

Yukarıdaki diyagramda tanımlanmış olan akış kapılarına ait akış özelliği Şekil 2-7’ te verilmiştir:



Şekil 2-7 SPI Akış Özellikleri

SysML diyagramı ile tanımlanmış olan sistemin elektronik devre şeması şeklinde olan alternatif gösterimi Şekil 2-8’ de verilmiştir:



Şekil 2-8 SPI Elektronik Devre Diyagramı

Diyagram gösterimlerinin karmaşıklığı açısından bakıldığında ise, SysML modeli, elektronik devre diyagramına karşılık çok daha basittir. İki gösterim arasındaki karşılaştırma Çizelge 2-2' de verilmiştir:

Çizelge 2-2 SPI için Diyagramlar Arası Karşılaştırma

	Düğüm Sayısı	**Kenar Sayısı**
SysML modeli	4	3
Elektronik Devre Şeması	3	27

Burada düğüm sayısı sistemde üst ve alt parçalara (iki diyagramda da kutu ile gösterilmiştir) karşılık gelmektedir. Örneğin, sspi, sspi_rx, sspi_tx, sspi_counter birer düğüm olarak ele alınmıştır. Kenarlar ise kutular arası veya dışarıya yapılan bağlantılar olarak sayılmıştır. Bununla birlikte, SysML diyagramında akış özelliği ile elektronik devre şemasındaki kutuların hazır olduğu varsayılmış olup sayıma katılmamıştır.

SysML modelinden geliştirilen tasarım için otomatik olarak üretilen ve elle yazılan satırlar Çizelge 2-3' te verilmiştir:

Çizelge 2-3 SPI için Üretilen ve Elle Yazılan VHDL Satırları (*)

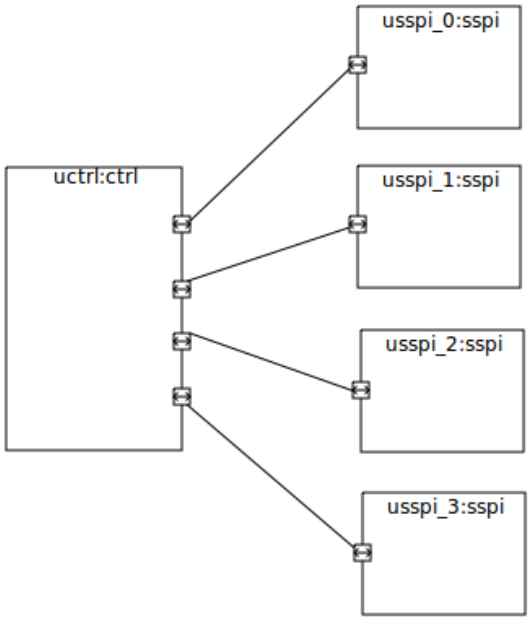
Dosya	**Üretilen VHDL Satır Sayısı**	**Elle Yazılan VHDL Satır Sayısı**
sspi.vhd	81	0
sspi_rx.vhd	15	38
sspi_tx.vhd	15	36
sspi_counter.vhd	15	11

(*) Sayıma boşluk ve yorum satırları dahil edilmemiştir.

Burada sspi.vhd tamamen otomatik olarak elde edilmiştir. Çünkü bu bileşen tamamen sistemin yapısını ifade ediyor olup sistemin davranışı ile ilgili bir bilgi içermemektedir. Üretilen VHDL tasarımları EK-A'da verilmiştir.

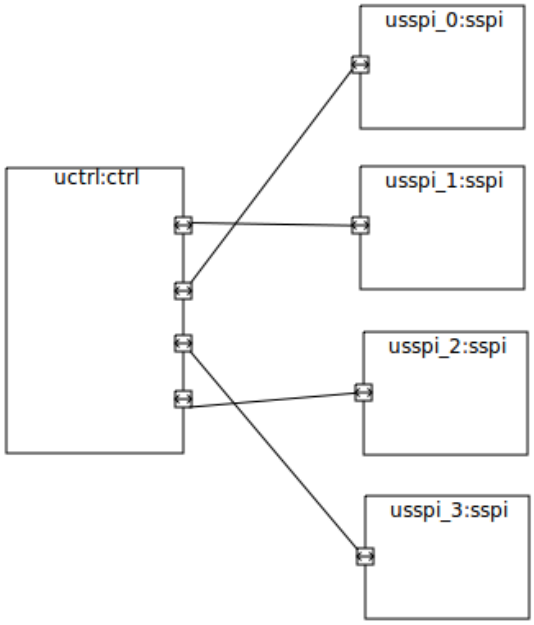
2.6.3 Örnek 3 – Sistemin Yapısal Modelinde Değişiklik Yapılması

SysML modellemenin kullanışlı olduğu durum için verilebilecek bir başka örnekte, yapısal modeli Şekil 2-9 'deki gibi olan bir sistemin yapısal modelinin Şekil 2-10'daki gibi değiştirilme ihtiyacının ortaya çıktığı durumlardır.



Şekil 2-9 Örnek-3 için Sistem Yapısal Modeli

Burada “usspi_0”, “usspi_1”, “usspi_2” ve “usspi_3” olguları (“instance”), bir önceki örnekte tanımlanmış olan “sspi” varlığından (“entity”) oluşturulmuştur. “uctrl” olgusu ise dört akış kapısı üzerinden “sspi” olgularına bağlanır. “uctrl” olgusu, “ctrl” isminde bir varlıktan oluşturulduğu varsayılmıştır. Sistemin yapısal modelinde yapılmış olan bağlantılar, Şekil 2-10’de gösterildiği şekilde değiştirilebilir:



Şekil 2-10 Örnek-3 için Sistemin Değiştirilmiş Yapısal Modeli

İki modelden üretilen VHDL tasarımı incelendiğinde her bir bağlantı değişikliği VHDL tasarımında yedi satırlık bir değişikliğe karşılık gelir. Yedi satırlık değişiklik "sspi_fs" akış özelliğinin (Şekil 2-7) içersinde yedi adet sinyal tanımlı olmasından kaynaklanır. Sonuç olarak bu da toplamda, SysML modeli üzerinde yapılmış olan dört bağlantı değişikliği için VHDL tasarımında yirmi sekiz satırlık bir değişikliğe karşılık gelir (EK-B).

2.7 Mevcut Yazılımlarla Karşılaştırma

UML ve SysML modelleme yapılabilen ticari ve açık kaynak yazılım araçları mevcuttur. Yazılımların özellikleri ile ilgili çeşitli kaynaklardan edinilen bilgilere göre değerlendirme yapılmıştır. İncelenen yazılımlar arasında öne çıkanlar aşağıda verilmiştir:

Enterprise Architect: Enterpise Architect, UML ağırlıklı model tabanlı geliştirme yazılımıdır. Sistem Mühendisleri için yazılımın özel bir sürümü satılmaktadır.

Enterprise Architect Systems Engineering Edition, gömülü sistem uygulamalarının geliştirilmesi için bir çok özellikler sunmaktadır. Bu özelliklerden biri

modelden VHDL tasarımı üretimidir. Yazılım, modelinden hem yapısal hemde davranışsal SysML modellerinden kod üretmektedir. (Rosenberg D. and Morella S.)

MagicDraw + SysML Plugin: Magic Draw daha çok genel amaçlı bir çizim aracıdır. SysML çizim özelliği haricen edinilen bir eklenti yazılımı ile gerçekleştirilmektedir. (No Magic Inc.)

Artisan Studio + SysML: Artisan Software, yine Enterprise Architect gibi UML ağırlıklı bir model tabanlı geliştirme yazılımıdır.

Kod üretimi için UML üzerine OMG tarafından tanımlanmış olan MARTE profilini kullanmaktadır.

UML 'den doğrudan VHDL tasarımı üretmek yerine daha üst seviyede bulunan SystemC dilinde çıktı üretir. Daha sonra kullanıcı SystemC çıktısından VHDL tasarımı üretir.

Bunun dışında üretici firmanın desteği ile yürütülen bir çalışmada, SysML modellerinden yine SystemC çıktısı üretilmesi amaçlanmaktadır. (Wythock, P.)

Embedded Plus SysML Toolkit: Rational Rose üzerinde çalışan eklenti yazılımdır. SysML 1.1 destekler ve VHDL üretimi konusunda doğrudan desteği yoktur. Bu yazılım, ClearCASE, ClearQuest, RequisitePro, Telelogic DOORS gibi ürün yaşam döngüsünde kullanılan diğer yazılım araçları ile de uyumlu çalışır. (EmbeddedPlus)

TOPCASED: SysML modellemenin yapılabileceği, en olgunlaşmış açık kaynak bir yazılım olup Eclipse yazılımına bir eklenti olarak sunulmaktadır. Yazılım tek açık kaynak örnek olması sebebi ile güncel dokümanları web sayfası üzerinde incelenmiştir. Bu yazılımda, VHDL üretimi konusunda doğrudan bir desteği yoktur ancak kullanıcı tarafından istenildiğinde, yazılıma eklenti yazılarak özellikler kazandırılabilmektedir (TOPCASED).

3 SONUÇ

Bu tez çalışması kapsamında, başlıca Elektronik Mühendisliği uygulamaları için, SysML modellemenin oluşturulabileceği ve oluşturulan bu yapısal model üzerinden otomatik VHDL tasarımın üretilebileceği açık kaynaklı bir yazılım geliştirilmiştir. Yazılımın şu anki hali ile sınırlı olarak SysML blok diyagramları ve iç blok diyagramları desteklenmiştir. VHDL tasarımının üretimi de bu diyagramlar üzerinden gerçekleştirilmiştir.

SysML blok diyagramları ve iç blok diyagramları, sistemin yapısal modelini ifade ettiğinden, otomatik olarak üretilen VHDL tasarımı da sistemin yapısal kısmını kapsamaktadır. İncelenen kullanım durumlarına ait örnekler incelendiğinde, sistemin yapısal kısmı tamamen otomatik olarak üretilebilmiştir.

Bunun ile birlikte diyagram karmaşıklığı açısından bakıldığında, SysML ile sayısal elektronik sistemlerin yapısal modellerinin, elektronik devre şemasına göre daha basit olarak ifade edilebildiği görülmüştür. Örnek olarak yonga üzeri sistemler (SoC), işlemci, haberleşme, hafıza denetçisi gibi bileşenleri bünyesinde barındırır. Bu bileşenlerin ara yüzlerinde de çok sayıda sinyal bulunabilir. Alt bileşenlerin çok olması başka bir anlamda sistemin yapısal modelinin de büyümesi anlamına gelir. Bu tip sistemlerin, elektronik devre şeması kullanarak tasarlanması, sistemi karışık ve zor anlaşılır hale getirir. Buna karşılık olarak, SysML ile, özellikle hiyerarşik ilişkili birden alt işlevsel bileşenlerden oluşan karmaşık büyük sistemlerin daha kolay modellenmesine olanak sağlanabilir.

SysML‘in kısa geçmişine rağmen, UML 2 tabanlı olması nedeni ile modellemenin yapılabileceği, ticari ve açık kaynak yazılımlar bulunmaktadır. Bunlardan Enterprise Architect Systems Engineering Edition ürününün, hem yapısal hem de davranışsal modelden otomatik olarak VHDL tasarımı üretimi yaptığı belirlenmiştir. Bu yazılımın, tez çalışması kapsamında edinilememiş olması nedeni ile yapısal modelden üretilen VHDL tasarımı için bir karşılaştırma yapılamamıştır.

Yazılım için, bir sonraki aşamada yapılabilecek olan geliştirmeler şu şekilde sıralanabilir:

1. Diğer SysML Diyagramları: Genel olarak bakıldığında tez çalışmasında geliştirilen yazılım şu an ki hali ile, modelleme konusunda, diğer incelenen yazılımlarla karşılaştırıldığında bir çok eksiği bulunmaktadır. Örnek olarak, hali hazırda bütün SysML diyagramları desteklenmemektedir. Eylem Diyagramları ve/veya Durum Diyagramları kullanılarak, sistemin davranışsal modeli oluşturulabilir ve bu modelden daha kapsamlı VHDL kodu üretimi gerçekleştirilebilir.
2. Tasarım Desteği: Özellikle Flip-flop devrelerinde, yarıkararlılık (metastability) sorunları için, tasarımın analiz edilip, gerekli olan çözümler otomatik olarak üretilebilir.
3. Model Kütüphanesi: Daha önce yapılmış olan tasarımların tekrar kullanılabilirliğini sağlamak amacı ile bir model kütüphanesi oluşturulabilir.
4. VHDL Kod Üretimi: SysML Modelinden VHDL kodu üretimi için başka bir yol, modelin standart olan XMI biçiminde bir veriye dönüştürülüp (genelde bir dosya olarak kaydedilerek) buradan ayrı bir yazılım ile dönüştürülen verinin işlenmesidir. Bu şekilde VHDL kodunu üretimi ayrı bir yazılım parçası olarak, SysML modelinin üretildiği grafik arayüzlü yazılımın dışına alınmış olur. Böylece modelin oluşturulması için, XMI biçimini destekleyen herhangi bir SysML modelleme yazılımı kullanılabilir.

4 KAYNAKLAR

- O.M.G., 2008, OMG Systems Modeling Language v1.1, Object Management Group, 234 p.
- O.M.G., 2009, OMG Unified Modeling Language Superstructure v2.2, Object Management Group, 724 p.
- Herzog, E. and Pandikow, A., 2005, SysML – an Assessment, International Council on Systems Engineering (INCOSE), 13 p.
- Hause, M., 2006, The SysML Modeling Language, International Council on Systems Engineering (INCOSE), 12 p.
- O.M.G., 2006, Meta Object Facility (MOF) Core Specification v2.0, Object Management Group, 75p.
- O.M.G., 2006, Diagram Interchange v1.0, Object Management Group, 76 p.
- Rosenberg D. and Morella S., Embedded Systems Development Using SysML, Sparx Systems Pty Ltd., 2010
- Wythock, P., UML/SysML-based Design Environment Is Based On Artisan Studio http://electronicdesign.com/content.aspx?topic=uml_sysml_based_design_environment_is_based_on_artisan_studio&catpath=news), Electronic Design Europe, 2010. En son erişim: 11.06.2010
- Artisan, Hardware, http://www.artisansoftwaretools.com/capabilities/lifecycle-support/hardware/, Artisan, En son erişim: 11.06.2010
- No Magic Inc. , http://www.magicdraw.com/sysml, No Magic Inc., 2010. En son erişim: 11.06.2010
- EmbeddedPlus, http://www.embeddedplus.com/SysML.php, EmbeddedPlus, 2010. En son erişim: 11.06.2010
- TOPCASED, TOPCASED The Open Source Toolkit for Critical Systems, http://www.topcased.org, TOPCASED, 2010. En son erişim: 11.06.2010

EKLER

EK-A. Örnek-2 VHDL Tasarımı

```
------------------ SSPI_COUNTER.VHD --------------------
entity sspi_counter
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
end sspi_counter;

architecture behavioural of sspi_counter

  signal counter : std_logic_vector(2 downto 0);

begin

  process (port_0_sspi_fs_reset, port_0_sspi_fs_sck) is
  begin
    if (port_0_sspi_fs_reset = '1') then
      counter <= "111";
    else
      if (port_0_sspi_fs_sck(CLK)) then
        counter <= counter + '1';
      end if;
    end if;
  end process;
```

```
  port_0_sspi_fs_rdy <= '1' when (counter = "111") else '0';

end behavioural;

-------------------- SSPI.VHD -------------------
entity sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
end sspi;

architecture behavioural of sspi
component sspi_rx
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
component sspi_tx
port
(
-- Interface definition for flow port port_0
```

```
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
component sspi_counter
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
begin

usspi_rx : sspi_rx
port map
(
port_0_sspi_fs_reset => port_0_sspi_fs_reset,
port_0_sspi_fs_sck => port_0_sspi_fs_sck,
port_0_sspi_fs_si => port_0_sspi_fs_si,
port_0_sspi_fs_so => port_0_sspi_fs_so,
port_0_sspi_fs_di => port_0_sspi_fs_di,
port_0_sspi_fs_do => port_0_sspi_fs_do,
port_0_sspi_fs_rdy => port_0_sspi_fs_rdy
);

usspi_tx : sspi_tx
port map
```

```
(
port_0_sspi_fs_reset => port_0_sspi_fs_reset,
port_0_sspi_fs_sck => port_0_sspi_fs_sck,
port_0_sspi_fs_si => port_0_sspi_fs_si,
port_0_sspi_fs_so => port_0_sspi_fs_so,
port_0_sspi_fs_di => port_0_sspi_fs_di,
port_0_sspi_fs_do => port_0_sspi_fs_do,
port_0_sspi_fs_rdy => port_0_sspi_fs_rdy
);

usspi_counter : sspi_counter
port map
(
port_0_sspi_fs_reset => port_0_sspi_fs_reset,
port_0_sspi_fs_sck => port_0_sspi_fs_sck,
port_0_sspi_fs_si => port_0_sspi_fs_si,
port_0_sspi_fs_so => port_0_sspi_fs_so,
port_0_sspi_fs_di => port_0_sspi_fs_di,
port_0_sspi_fs_do => port_0_sspi_fs_do,
port_0_sspi_fs_rdy => port_0_sspi_fs_rdy
);
end behavioural;

---------------------- SSPI_RX.VHD ------------------------

entity sspi_rx
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
```

```
);
end sspi_rx;

architecture behavioural of sspi_rx

  type input_t is
  record
    -- inputs
    si        : std_logic;
  end record;

  type register_t is
  record
    -- registers
    sh_reg    : std_logic_vector(7 downto 0);
  end record;

  function defaults return register_t is
    variable v : register_t;
  begin
    v.sh_reg := X"00";
    return v;
  end function;

  signal r   : register_t := defaults;
  signal rin : register_t;
  signal i   : input_t;

begin

  i.si <= port_0_sspi_fs_si;

  cmb : process (r, i) is
    variable v : register_t;
  begin
    v := r;
```

```
    v.sh_reg(7 downto 0)  := v.sh_reg(6 downto 0) & i.si;
    rin <= v;
  end process;

  seq : process (port_0_sspi_fs_reset, port_0_sspi_fs_sck) is
  begin
    if (port_0_sspi_fs_reset = '1') then
      r <= defaults;
    else
      if (RISING_EDGE(port_0_sspi_fs_sck)) then
        r <= rin;
      end if;
    end if;
  end process;

  port_0_sspi_fs_do <= r.sh_reg;

end behavioural;

---------------------- SSPI_TX.VHD ---------------------------

entity sspi_tx
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector;
port_0_sspi_fs_do : out std_logic_vector;
port_0_sspi_fs_rdy : out std_logic;
);
end sspi_tx;

architecture behavioural of sspi_tx
```

```
  type input_t is
  record
    -- inputs
    si        : std_logic;
  end record;

  type register_t is
  record
    sh_reg    : std_logic_vector(7 downto 0);
  end record;

  function defaults return register_t is
    variable v : register_t;
  begin
    v.sh_reg := X"00";
    return v;
  end function;

  signal r   : register_t := defaults;
  signal rin : register_t;
  signal i   : input_t;

begin

  i.di <= port_0_sspi_fs_di;

  cmb : process (r, i) is
    variable v : register_t;
  begin
    v := r;
    if (port_0_sspi_fs_rdy = '1') then
      v.sh_reg(7 downto 0) := i.di;
    else
      v.sh_reg(7 downto 0) := v.sh_reg(6 downto 0) & '0';
    end if;
```

```
    rin <= v;
  end process;

  seq : process (port_0_sspi_fs_reset, port_0_sspi_fs_sck) is
  begin
    if (port_0_sspi_fs_reset = '1') then
      r <= defaults;
    else
      if (FALLING_EDGE(port_0_sspi_fs_sck)) then
        r <= rin;
      end if;
    end if;
  end process;

  port_0_sspi_fs_so  <= r.sh_reg(7);

end behavioural;
```

EK-B. Örnek-3 VHDL Tasarımı

```
---------------------- SYS_MC_SPI.VHD --------------------
-- DEĞİŞİKLİK ÖNCESİ
entity sys_mc_spi
end sys_mc_spi;

architecture behavioural of sys_mc_spi
component sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
```

```
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component ctrl
port
(
-- Interface definition for flow port port_a
port_a_sspi_fs_reset : in std_logic;
port_a_sspi_fs_sck : in std_logic;
port_a_sspi_fs_si : in std_logic;
port_a_sspi_fs_so : out std_logic;
port_a_sspi_fs_di : in std_logic_vector(7 downto 0);
port_a_sspi_fs_do : out std_logic_vector(7 downto 0);
port_a_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_b
port_b_sspi_fs_reset : in std_logic;
port_b_sspi_fs_sck : in std_logic;
port_b_sspi_fs_si : in std_logic;
```

```
port_b_sspi_fs_so : out std_logic;
port_b_sspi_fs_di : in std_logic_vector(7 downto 0);
port_b_sspi_fs_do : out std_logic_vector(7 downto 0);
port_b_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_c
port_c_sspi_fs_reset : in std_logic;
port_c_sspi_fs_sck : in std_logic;
port_c_sspi_fs_si : in std_logic;
port_c_sspi_fs_so : out std_logic;
port_c_sspi_fs_di : in std_logic_vector(7 downto 0);
port_c_sspi_fs_do : out std_logic_vector(7 downto 0);
port_c_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_d
port_d_sspi_fs_reset : in std_logic;
port_d_sspi_fs_sck : in std_logic;
port_d_sspi_fs_si : in std_logic;
port_d_sspi_fs_so : out std_logic;
port_d_sspi_fs_di : in std_logic_vector(7 downto 0);
port_d_sspi_fs_do : out std_logic_vector(7 downto 0);
port_d_sspi_fs_rdy : out std_logic;
);

signal c_a_0_uctrl_port_a_usspi_0_port_0_reset : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_sck : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_si : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_so : out std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_di : in std_logic_vector(7
downto 0);
signal c_a_0_uctrl_port_a_usspi_0_port_0_do : out std_logic_vector(7
downto 0);
signal c_a_0_uctrl_port_a_usspi_0_port_0_rdy : out std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_reset : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_sck : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_si : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_so : out std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_di : in std_logic_vector(7
downto 0);
```

```
signal c_b_1_uctrl_port_b_usspi_1_port_0_do : out std_logic_vector(7
downto 0);
signal c_b_1_uctrl_port_b_usspi_1_port_0_rdy : out std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_reset : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_sck : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_si : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_so : out std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_di : in std_logic_vector(7
downto 0);
signal c_c_2_uctrl_port_c_usspi_2_port_0_do : out std_logic_vector(7
downto 0);
signal c_c_2_uctrl_port_c_usspi_2_port_0_rdy : out std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_reset : in std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_sck : in std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_si : in std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_so : out std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_di : in std_logic_vector(7
downto 0);
signal c_d_3_uctrl_port_d_usspi_3_port_0_do : out std_logic_vector(7
downto 0);
signal c_d_3_uctrl_port_d_usspi_3_port_0_rdy : out std_logic;
begin
usspi_0 : sspi
port map
(
port_0_sspi_fs_reset => c_a_0_uctrl_port_a_usspi_0_port_0_reset,
port_0_sspi_fs_sck => c_a_0_uctrl_port_a_usspi_0_port_0_sck,
port_0_sspi_fs_si => c_a_0_uctrl_port_a_usspi_0_port_0_si,
port_0_sspi_fs_so => c_a_0_uctrl_port_a_usspi_0_port_0_so,
port_0_sspi_fs_di => c_a_0_uctrl_port_a_usspi_0_port_0_di,
port_0_sspi_fs_do => c_a_0_uctrl_port_a_usspi_0_port_0_do,
port_0_sspi_fs_rdy => c_a_0_uctrl_port_a_usspi_0_port_0_rdy
);
usspi_1 : sspi
port map
(
port_0_sspi_fs_reset => c_b_1_uctrl_port_b_usspi_1_port_0_reset,
port_0_sspi_fs_sck => c_b_1_uctrl_port_b_usspi_1_port_0_sck,
```

```
port_0_sspi_fs_si => c_b_1_uctrl_port_b_usspi_1_port_0_si,
port_0_sspi_fs_so => c_b_1_uctrl_port_b_usspi_1_port_0_so,
port_0_sspi_fs_di => c_b_1_uctrl_port_b_usspi_1_port_0_di,
port_0_sspi_fs_do => c_b_1_uctrl_port_b_usspi_1_port_0_do,
port_0_sspi_fs_rdy => c_b_1_uctrl_port_b_usspi_1_port_0_rdy
);
usspi_2 : sspi
port map
(
port_0_sspi_fs_reset => c_c_2_uctrl_port_c_usspi_2_port_0_reset,
port_0_sspi_fs_sck => c_c_2_uctrl_port_c_usspi_2_port_0_sck,
port_0_sspi_fs_si => c_c_2_uctrl_port_c_usspi_2_port_0_si,
port_0_sspi_fs_so => c_c_2_uctrl_port_c_usspi_2_port_0_so,
port_0_sspi_fs_di => c_c_2_uctrl_port_c_usspi_2_port_0_di,
port_0_sspi_fs_do => c_c_2_uctrl_port_c_usspi_2_port_0_do,
port_0_sspi_fs_rdy => c_c_2_uctrl_port_c_usspi_2_port_0_rdy
);
usspi_3 : sspi
port map
(
port_0_sspi_fs_reset => c_d_3_uctrl_port_d_usspi_3_port_0_reset,
port_0_sspi_fs_sck => c_d_3_uctrl_port_d_usspi_3_port_0_sck,
port_0_sspi_fs_si => c_d_3_uctrl_port_d_usspi_3_port_0_si,
port_0_sspi_fs_so => c_d_3_uctrl_port_d_usspi_3_port_0_so,
port_0_sspi_fs_di => c_d_3_uctrl_port_d_usspi_3_port_0_di,
port_0_sspi_fs_do => c_d_3_uctrl_port_d_usspi_3_port_0_do,
port_0_sspi_fs_rdy => c_d_3_uctrl_port_d_usspi_3_port_0_rdy
);
uctrl : ctrl
port map
(
  port_a_sspi_fs_reset => c_a_0_uctrl_port_a_usspi_0_port_0_reset,
  port_a_sspi_fs_sck => c_a_0_uctrl_port_a_usspi_0_port_0_sck,
  port_a_sspi_fs_si => c_a_0_uctrl_port_a_usspi_0_port_0_si,
  port_a_sspi_fs_so => c_a_0_uctrl_port_a_usspi_0_port_0_so,
  port_a_sspi_fs_di => c_a_0_uctrl_port_a_usspi_0_port_0_di,
```

```
  port_a_sspi_fs_do => c_a_0_uctrl_port_a_usspi_0_port_0_do,
  port_a_sspi_fs_rdy => c_a_0_uctrl_port_a_usspi_0_port_0_rdy,
  port_b_sspi_fs_reset => c_b_1_uctrl_port_b_usspi_1_port_0_reset,
  port_b_sspi_fs_sck => c_b_1_uctrl_port_b_usspi_1_port_0_sck,
  port_b_sspi_fs_si => c_b_1_uctrl_port_b_usspi_1_port_0_si,
  port_b_sspi_fs_so => c_b_1_uctrl_port_b_usspi_1_port_0_so,
  port_b_sspi_fs_di => c_b_1_uctrl_port_b_usspi_1_port_0_di,
  port_b_sspi_fs_do => c_b_1_uctrl_port_b_usspi_1_port_0_do,
  port_b_sspi_fs_rdy => c_b_1_uctrl_port_b_usspi_1_port_0_rdy,
  port_c_sspi_fs_reset => c_c_2_uctrl_port_c_usspi_2_port_0_reset,
  port_c_sspi_fs_sck => c_c_2_uctrl_port_c_usspi_2_port_0_sck,
  port_c_sspi_fs_si => c_c_2_uctrl_port_c_usspi_2_port_0_si,
  port_c_sspi_fs_so => c_c_2_uctrl_port_c_usspi_2_port_0_so,
  port_c_sspi_fs_di => c_c_2_uctrl_port_c_usspi_2_port_0_di,
  port_c_sspi_fs_do => c_c_2_uctrl_port_c_usspi_2_port_0_do,
  port_c_sspi_fs_rdy => c_c_2_uctrl_port_c_usspi_2_port_0_rdy,
  port_d_sspi_fs_reset => c_d_3_uctrl_port_d_usspi_3_port_0_reset,
  port_d_sspi_fs_sck => c_d_3_uctrl_port_d_usspi_3_port_0_sck,
  port_d_sspi_fs_si => c_d_3_uctrl_port_d_usspi_3_port_0_si,
  port_d_sspi_fs_so => c_d_3_uctrl_port_d_usspi_3_port_0_so,
  port_d_sspi_fs_di => c_d_3_uctrl_port_d_usspi_3_port_0_di,
  port_d_sspi_fs_do => c_d_3_uctrl_port_d_usspi_3_port_0_do,
  port_d_sspi_fs_rdy => c_d_3_uctrl_port_d_usspi_3_port_0_rdy
);
end behavioural;

---------------------- SYS_MC_SPI_2.VHD --------------------
-- DEĞİŞİKLİK SONRASI
entity sys_mc_spi_2
end sys_mc_spi_2;

architecture behavioural of sys_mc_spi_2
component sspi
port
(
-- Interface definition for flow port port_0
```

```
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
-- Interface definition for flow port port_0
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component sspi
port
(
-- Interface definition for flow port port_0
```

```
port_0_sspi_fs_reset : in std_logic;
port_0_sspi_fs_sck : in std_logic;
port_0_sspi_fs_si : in std_logic;
port_0_sspi_fs_so : out std_logic;
port_0_sspi_fs_di : in std_logic_vector(7 downto 0);
port_0_sspi_fs_do : out std_logic_vector(7 downto 0);
port_0_sspi_fs_rdy : out std_logic;
);
component ctrl
port
(
-- Interface definition for flow port port_a
port_a_sspi_fs_reset : in std_logic;
port_a_sspi_fs_sck : in std_logic;
port_a_sspi_fs_si : in std_logic;
port_a_sspi_fs_so : out std_logic;
port_a_sspi_fs_di : in std_logic_vector(7 downto 0);
port_a_sspi_fs_do : out std_logic_vector(7 downto 0);
port_a_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_b
port_b_sspi_fs_reset : in std_logic;
port_b_sspi_fs_sck : in std_logic;
port_b_sspi_fs_si : in std_logic;
port_b_sspi_fs_so : out std_logic;
port_b_sspi_fs_di : in std_logic_vector(7 downto 0);
port_b_sspi_fs_do : out std_logic_vector(7 downto 0);
port_b_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_c
port_c_sspi_fs_reset : in std_logic;
port_c_sspi_fs_sck : in std_logic;
port_c_sspi_fs_si : in std_logic;
port_c_sspi_fs_so : out std_logic;
port_c_sspi_fs_di : in std_logic_vector(7 downto 0);
port_c_sspi_fs_do : out std_logic_vector(7 downto 0);
port_c_sspi_fs_rdy : out std_logic;
-- Interface definition for flow port port_d
```

```
port_d_sspi_fs_reset : in std_logic;
port_d_sspi_fs_sck : in std_logic;
port_d_sspi_fs_si : in std_logic;
port_d_sspi_fs_so : out std_logic;
port_d_sspi_fs_di : in std_logic_vector(7 downto 0);
port_d_sspi_fs_do : out std_logic_vector(7 downto 0);
port_d_sspi_fs_rdy : out std_logic;
);

signal c_a_0_uctrl_port_a_usspi_0_port_0_reset : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_sck : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_si : in std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_so : out std_logic;
signal c_a_0_uctrl_port_a_usspi_0_port_0_di : in std_logic_vector(7
downto 0);
signal c_a_0_uctrl_port_a_usspi_0_port_0_do : out std_logic_vector(7
downto 0);
signal c_a_0_uctrl_port_a_usspi_0_port_0_rdy : out std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_reset : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_sck : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_si : in std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_so : out std_logic;
signal c_b_1_uctrl_port_b_usspi_1_port_0_di : in std_logic_vector(7
downto 0);
signal c_b_1_uctrl_port_b_usspi_1_port_0_do : out std_logic_vector(7
downto 0);
signal c_b_1_uctrl_port_b_usspi_1_port_0_rdy : out std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_reset : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_sck : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_si : in std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_so : out std_logic;
signal c_c_2_uctrl_port_c_usspi_2_port_0_di : in std_logic_vector(7
downto 0);
signal c_c_2_uctrl_port_c_usspi_2_port_0_do : out std_logic_vector(7
downto 0);
signal c_c_2_uctrl_port_c_usspi_2_port_0_rdy : out std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_reset : in std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_sck : in std_logic;
```

```
signal c_d_3_uctrl_port_d_usspi_3_port_0_si : in std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_so : out std_logic;
signal c_d_3_uctrl_port_d_usspi_3_port_0_di : in std_logic_vector(7
downto 0);
signal c_d_3_uctrl_port_d_usspi_3_port_0_do : out std_logic_vector(7
downto 0);
signal c_d_3_uctrl_port_d_usspi_3_port_0_rdy : out std_logic;
begin
usspi_0 : sspi
port map
(
port_0_sspi_fs_reset => c_b_1_uctrl_port_b_usspi_1_port_0_reset,
port_0_sspi_fs_sck => c_b_1_uctrl_port_b_usspi_1_port_0_sck,
port_0_sspi_fs_si => c_b_1_uctrl_port_b_usspi_1_port_0_si,
port_0_sspi_fs_so => c_b_1_uctrl_port_b_usspi_1_port_0_so,
port_0_sspi_fs_di => c_b_1_uctrl_port_b_usspi_1_port_0_di,
port_0_sspi_fs_do => c_b_1_uctrl_port_b_usspi_1_port_0_do,
port_0_sspi_fs_rdy => c_b_1_uctrl_port_b_usspi_1_port_0_rdy
);
usspi_1 : sspi
port map
(
port_0_sspi_fs_reset => c_a_0_uctrl_port_a_usspi_0_port_0_reset,
port_0_sspi_fs_sck => c_a_0_uctrl_port_a_usspi_0_port_0_sck,
port_0_sspi_fs_si => c_a_0_uctrl_port_a_usspi_0_port_0_si,
port_0_sspi_fs_so => c_a_0_uctrl_port_a_usspi_0_port_0_so,
port_0_sspi_fs_di => c_a_0_uctrl_port_a_usspi_0_port_0_di,
port_0_sspi_fs_do => c_a_0_uctrl_port_a_usspi_0_port_0_do,
port_0_sspi_fs_rdy => c_a_0_uctrl_port_a_usspi_0_port_0_rdy
);
usspi_2 : sspi
port map
(
port_0_sspi_fs_reset => c_d_3_uctrl_port_d_usspi_3_port_0_reset,
port_0_sspi_fs_sck => c_d_3_uctrl_port_d_usspi_3_port_0_sck,
port_0_sspi_fs_si => c_d_3_uctrl_port_d_usspi_3_port_0_si,
port_0_sspi_fs_so => c_d_3_uctrl_port_d_usspi_3_port_0_so,
```

```
port_0_sspi_fs_di => c_d_3_uctrl_port_d_usspi_3_port_0_di,
port_0_sspi_fs_do => c_d_3_uctrl_port_d_usspi_3_port_0_do,
port_0_sspi_fs_rdy => c_d_3_uctrl_port_d_usspi_3_port_0_rdy
);
usspi_3 : sspi
port map
(
port_0_sspi_fs_reset => c_c_2_uctrl_port_c_usspi_2_port_0_reset,
port_0_sspi_fs_sck => c_c_2_uctrl_port_c_usspi_2_port_0_sck,
port_0_sspi_fs_si => c_c_2_uctrl_port_c_usspi_2_port_0_si,
port_0_sspi_fs_so => c_c_2_uctrl_port_c_usspi_2_port_0_so,
port_0_sspi_fs_di => c_c_2_uctrl_port_c_usspi_2_port_0_di,
port_0_sspi_fs_do => c_c_2_uctrl_port_c_usspi_2_port_0_do,
port_0_sspi_fs_rdy => c_c_2_uctrl_port_c_usspi_2_port_0_rdy
);
uctrl : ctrl
port map
(
  port_a_sspi_fs_reset => c_a_0_uctrl_port_a_usspi_0_port_0_reset,
  port_a_sspi_fs_sck => c_a_0_uctrl_port_a_usspi_0_port_0_sck,
  port_a_sspi_fs_si => c_a_0_uctrl_port_a_usspi_0_port_0_si,
  port_a_sspi_fs_so => c_a_0_uctrl_port_a_usspi_0_port_0_so,
  port_a_sspi_fs_di => c_a_0_uctrl_port_a_usspi_0_port_0_di,
  port_a_sspi_fs_do => c_a_0_uctrl_port_a_usspi_0_port_0_do,
  port_a_sspi_fs_rdy => c_a_0_uctrl_port_a_usspi_0_port_0_rdy,
  port_b_sspi_fs_reset => c_b_1_uctrl_port_b_usspi_1_port_0_reset,
  port_b_sspi_fs_sck => c_b_1_uctrl_port_b_usspi_1_port_0_sck,
  port_b_sspi_fs_si => c_b_1_uctrl_port_b_usspi_1_port_0_si,
  port_b_sspi_fs_so => c_b_1_uctrl_port_b_usspi_1_port_0_so,
  port_b_sspi_fs_di => c_b_1_uctrl_port_b_usspi_1_port_0_di,
  port_b_sspi_fs_do => c_b_1_uctrl_port_b_usspi_1_port_0_do,
  port_b_sspi_fs_rdy => c_b_1_uctrl_port_b_usspi_1_port_0_rdy,
  port_c_sspi_fs_reset => c_c_2_uctrl_port_c_usspi_2_port_0_reset,
  port_c_sspi_fs_sck => c_c_2_uctrl_port_c_usspi_2_port_0_sck,
  port_c_sspi_fs_si => c_c_2_uctrl_port_c_usspi_2_port_0_si,
  port_c_sspi_fs_so => c_c_2_uctrl_port_c_usspi_2_port_0_so,
```

```
  port_c_sspi_fs_di => c_c_2_uctrl_port_c_usspi_2_port_0_di,
  port_c_sspi_fs_do => c_c_2_uctrl_port_c_usspi_2_port_0_do,
  port_c_sspi_fs_rdy => c_c_2_uctrl_port_c_usspi_2_port_0_rdy,
  port_d_sspi_fs_reset => c_d_3_uctrl_port_d_usspi_3_port_0_reset,
  port_d_sspi_fs_sck => c_d_3_uctrl_port_d_usspi_3_port_0_sck,
  port_d_sspi_fs_si => c_d_3_uctrl_port_d_usspi_3_port_0_si,
  port_d_sspi_fs_so => c_d_3_uctrl_port_d_usspi_3_port_0_so,
  port_d_sspi_fs_di => c_d_3_uctrl_port_d_usspi_3_port_0_di,
  port_d_sspi_fs_do => c_d_3_uctrl_port_d_usspi_3_port_0_do,
  port_d_sspi_fs_rdy => c_d_3_uctrl_port_d_usspi_3_port_0_rdy
);
end behavioural;
```