Computer Aided Implementation using Xilinx System Generator

Examensarbete utfört i elektroniksystem
vid Linköpings tekniska högskola

av

Henrik Eriksson

LITH-ISY-EX-3451-2004

Linköping 2004

Computer Aided Implementation using Xilinx System Generator

Examensarbete utfört i elektroniksystem
vid Linköpings tekniska högskola

av
Henrik Eriksson

LITH-ISY-EX-3451-2004

Handledare:
Morgan Andersson, Ericsson Microwave Systems AB

Examinator:
Kent Palmkvist, ISY

Linköping, Januari 29, 2004

LINKÖPINGS UNIVERSITET

Avdelning, Institution
Division, Department

Institutionen för systemteknik
581 83 LINKÖPING

Datum
Date
2004-01-29

Språk
Language
X Svenska/Swedish
Engelska/English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats

Övrig rapport

ISBN

ISRN LITH-ISY-EX-3451-2004

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2004/3451/

Titel
Title
Datorstödd implementering med hjälp av Xilinx System Generator

Computer Aided Implementation using Xilinx System Generator

Författare
Author
Henrik Eriksson

Sammanfattning
Abstract
The development in electronics increases the demand for good design methods and design tools in the field of electrical engeneering. To improve their design methods Ericsson Microwave Systems AB is interested in using computer tools to create a link between the specification and the implementation of a digital system in a FPGA.

Xilinx System Generator for DSP is a tool for implementing a model of a digital signalprocessing algorithm in a Xilinx FPGA. To evaluate Xilinx System Generator two testcases has been designed. The testcases are selected to represent the FPGA designs made at Ericsson Microwave Systems.

The testcases show that Xilinx System Generator can be used to effectivly implement a model made in Simulink in a FPGA from Xilinx. The result of the implementation is comparable to the implementation of VHDL code written by hand.

The use of tools for implementation of a model in hardware cause change in the design methods used at Ericsson Microwave Systems. The higher level of abstraction introduced by System Generator results in the design decisions made at system level having a higher impact on the final realization.

Nyckelord
Keyword
FPGA, VHDL, Xilinx System Generator, Simulink, Design tools

Abstract

The development in electronics increases the demand for good design methods and design tools in the field of electrical engeneering. To improve their design methods Ericsson Microwave Systems AB is interested in using computer tools to create a link between the specification and the implementation of a digital system in a FPGA.

Xilinx System Generator for DSP is a tool for implementing a model of a digital signalprocessing algorithm in a Xilinx FPGA. To evaluate Xilinx System Generator two testcases has been designed. The testcases are selected to represent the FPGA designs made at Ericsson Microwave Systems.

The testcases show that Xilinx System Generator can be used to effectivly implement a model made in Simulink in a FPGA from Xilinx. The result of the implementation is comparable to the implementation of VHDL code written by hand.

The use of tools for implementation of a model in hardware cause change in the design methods used at Ericsson Microwave Systems. The higher level of abstraction introduced by System Generator results in the design decisions made at system level having a higher impact on the final realization.

Sammanfattning

Utvecklingen inom elektronik gör att allt högre krav ställs på de metoder och datorverktyg som används inom elektronikkonstruktion. För att förbättra konstruktionsprocessen vid delssytemkonstruktion är man på Ericsson Microwave Systems AB intresserade av att använda datorverktyg för att skapa en direktkoppling mellan en specifikation och den slutliga implementationen av en digital funktion i en FPGA.

Verktyget Xilinx System Generator for DSP erbjuder möjlighet att implementera en modell av en digital signalbehandlingsfunktion i en Xilinx FPGA. För att utvärdera Xilinx System Generator har två testfall konstruerats. Testfallen är utvalda för att representera de funktioner som ingår i de system EMW producerar.

Testfallen visar att Xilinx System Generator kan användas för att effektivt implementera en modell gjord i Simulink i en FPGA från Xilinx. Resultatet av implementationen är jämförbart med det resultat som den traditionella metoden med handskriven VHDL kod ger.

Användning av verktyg för direktimplementation av en modell i hårdvara medför förändringar i de arbetsmetoder som används vid delsystemkonstruktion på Ericsson Microwave Systems. Den högre abstraktionsnivå som användning av System Generator medför innebär att de beslut som fattas på delsystemnivå får större betydelse för den slutliga realiseringen.

Förord

Jag skulle vilja tacka alla på Ericsson Microwave Systems som hjälpt mig under arbetet. Speciellt skulle jag vilja tacka min handledare Morgan Andersson och Fernando Kjellén för all hjälp och många givande diskussioner.

Förkortningar

ASIC	Application Specific Integrated Circuit
AGC	Automatic Gain Control
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
DAC	Digital to Analog Converter
DCM	Digital Clock Manager
DSP	Digital Signal Processing
EDIF	Electronic Design Interchange Format
EMW	Ericsson Microwave Systems
FDA	Filter Design & Analysis
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
I/O	Input/Output
IF	Intermediate Frequency
IP	Intellectual Property
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look Up Table
MAC	Multiply Accumulate
MSB	Most Significant Bit
NGD	Native Generic Database
PAR	Place And Route
PCI	Peripheral Component Interconnect
PLL	Phase Locked Loop
PRI	Pulsrepetitionsintervall
RAM	Random Access Memory
RF	Radio Frequency
RTL	Register Transfer Level
SoC	System on Chip
SRAM	Static Random Access Memory
SSRAM	Synchronous Static Random Access Memory
USB	Universal Serial Bus
VHDL	Very high speed integrated circuit Hardware Description Language

Innehållsförteckning

Figurförteckning

1 Inledning

Vid elektronikutveckling ställs allt högre krav på hög prestanda och kort utvecklingstid. För att klara de ökande kraven ställs därför högre krav på de datorverktyg som används vid konstruktion och implementation, [1].

Vid systemkonstruktion kan simuleringsverktyg användas för att simulera systemet. Utvecklingen av verktyg för att implementera en simuleringsmodell i t ex en FPGA eller digital signal processor har dessutom gjort dessa simuleringsverktyg till ett alternativ vid konstruktion.

MATLAB och Simulink är två verktyg från MathWorks, Inc som har funktioner för att implementera en simuleringsmodell i en FPGA eller som mjukvara i en digital signal processor.

1.1 Bakgrund

Ericsson Microwave Systems AB är ett världsledande företag inom försvarselektronik och militära informationsnätverk. Ericsson Microwave Systems utvecklar lösningar för att erbjuda kunden informationsövertag. Företaget erbjuder utveckling och produktion av framförallt radarsensorer men också nätverkslösningar för informationsöverföring, [2].

De produkter som Ericsson Microwave Systems producerar ingår i mycket avancerade tekniska system. Höga krav ställs därför på hög prestanda, säkerhet och kvalite.

För att förbättra konstruktionsprocessen är man på Ericsson Microwave Systems intresserade av att använda datorverktyg för att skapa en direktkoppling mellan en teknisk specifikation och den slutliga implementationen.

Arbetet har utförts på enheten FX/HX inom Ericsson Microwave Systems som arbetar med delsystemdesign inom antenn och mikrovågsteknik.

1.2 Syfte

Syftet med arbetet är att utvärdera möjligheten att gå från en modell av en digital funktion i MATLAB/Simulink direkt till en implementation i en Xilinx FPGA.

Ett verktyg som implementerar en modell gjord i Simulink i en Xilinx FPGA är Xilinx System Generator for DSP. För utvärdering av System Generator finns färdiga hårdvaruplattformar från Xilinx. Ett delmål i arbetet har varit att undersöka om en plattform av det slag som Xilinx erbjuder kan användas vid utvecklingen av demonstratorer och prototyper på EMW.

En diskussion kring vad datorstödd implementation med hjälp av Simulink och System Generator innebär för arbetet med delsystemkonstruktion på Ericsson Microwave Systems ingår också i arbetet.

1.3 Metod

För att utvärdera Xilinx System Generator och möjligheten att impementera en Simulinkmodell direkt i en Xilinx FPGA har följande frågor besvarats:

- Kan Simulink användas för att modellera de system som EMW konstruerar?
- Vilka kunskaper om VHDL och digitalkonstruktion krävs för att använda System Generator?
- Är den realisering som erhålls med hjälp av Xilinx System Generator jämförbar med en realisering som fås vid traditionell VHDL konstruktion?

För att besvara dessa frågor har två testfall konstruerats och implementerats och resultatet studerats. Testfallen är utvalda för att representera de konstruktioner som görs i FPGA:er på Ericsson Microwave Systems.

Testfallen ger dessutom svar på frågan om en generell hårdvaruplattform kan användas för utveckling av digitala funktioner på EMW.

1.4 Läsanvisning

Rapporten riktar sig till läsare som har viss erfarenhet av elektroteknik och digital signalbehandling. Föklaringar till grundläggande begrepp och termer ges inte i rapporten.

Kapitel 1 ger bakgrund, presenterar syftet och beskriver metoden för examensarbetet.

Kapitel 2-4 ger läsare som saknar erfarenhet av MATLAB, Simulink och FPGA användning en introduktion till dessa områden. Dessa kapitel introducerar dessutom många av de begrepp som används i rapporten.

Kapitel 5 och 6 beskriver vektyget Xilinx System Generator som använts under arbetet respektive den hårdvara som använts.

Kapitel 7 och 8 beskriver de testfall som gjorts för att utvärdera verktygen och plattformen Xtreme DSP.

Kapitel 9 behandlar användning av datorstöd i allmänhet och Simulink och System Generator i synnerhet. En metod för dokumentation av en Simulinkmodell presenteras.

Kapitel 10 presenterar resultatet av arbetet.

Kaptiel 11 innehåller referenser till de källor som använts under arbetet

Två bilagor som innehåller källkod till de funktioner som används vid utbyte av information mellan Word och Simulink avslutar rapporten.

2 MATLAB

MATLAB är ett programpaket för tekniska beräkningar som används inom vitt skilda områden som t ex signal- och bildbehandling, reglerteknik, bioteknik och ekonomi. MATLAB används idag av över 500 000 användare över hela världen, [3].

2.1 Historia

Under mitten av 70-talet utvecklades LINPACK och EISPACK två programbibliotek för att lösa linjära ekvationer respektive egenvärdesproblem. I slutet av 70-talet ville Cleve Moler, då vid University of New Mexico, använda LINPACK och EISPACK i sin undervisning i linjär algebra. Problemet var att LINPACK och EISPACK var skrivna i FORTRAN och Moler tyckte inte att studenterna skulle behöva lära sig FORTRAN för att delta i hans kurs. Han började därför på egen hand utveckla ett program som enkelt skulle ge användaren tillgång till de kraftfulla beräkningsfunktionerna i LINPACK och EISPACK. Han döpte sitt program MATLAB, som står för MATrix LABoratory.

MATLAB spreds under slutet av 70- och början av 80-talet i universitetsvärlden och 1983 träffade Moler John Little vid ett besök på Stanford University. Little var ingenjör och insåg snabbt möjligheterna med MATLAB. Little och Moler började utveckla en kommersiell version av MATLAB som bland annat innehöll grafikmöjligheter. 1984 grundades MathWorks, Inc. för att marknadsföra och utveckla MATLAB, [4].

2.2 Användning

MATLAB innehåller över 600 inbyggda funktioner som ger användaren tillgång till högpresterande numeriska beräkningar. Matematiken är optimerad för matris- och vektorberäkningar och de funktioner som används bygger på välutvecklade programbibliotek som t ex LAPACK.

För att ytterligare utöka möjligheterna med MATLAB finns ett flertal verktygslådor (eng. toolbox) som innehåller utökade funktioner inom specifika användningsområden. Exempel på verktygslådor från MathWorks, Inc. är Signal Processing Toolbox, Communication Toolbox, Optimization Toolbox och Financial Toolbox, [3].

MATLAB har utvecklats till något av en industristandard för tekniska beräkningar inom många områden och många tredjepartstillverkare utvecklar egna verktygslådor och tillägg till MATLAB.

3 Simulink

Simulink är ett programpaket för modellering och simulering av dynamiska system, som utvecklas av Mathworks, Inc. Simulink har stöd för linjära såväl som olinjära system och modellering med kontinuerlig tidskala, samplade system eller hybrida system som blandar samplad och kontinuerlig tidskala, [5].

Detta kapitel behandlar Simulink generellt och som simuleringsverktyg. Xilinx System Generator for DSP är ett verktyg som dessutom möjliggör att Simulink används för konstruktion av digitala funktioner. System Generator beskrivs i kapitel 5.

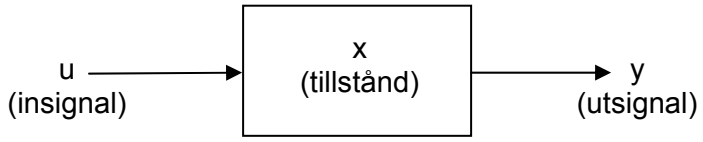
3.1 Modellering

En modell i Simulink byggs upp genom ett grafiskt användargränssnitt som ett blockdiagram. Blockdiagrammet består av block och ledningar. Varje block motsvarar ett dynamiskt system som Simulink vet hur det skall simuleras. Ledningarna motsvarar kommunikation mellan olika block.

Simulink innehåller ett blockbibliotek som kan användas för att bygga modeller av olika typer av system. Ytterligare block finns att tillgå genom så kallade blocksets som är Simulinks motsvarighet till MATLABs verktygslådor. Exempel på blocksets från MathWorks, Inc. är DSP blockset, Communication blockset och Fixed-Point blockset.

3.1.1 Egenskaper för Simulink block

Ett block i Simulink representerar ett grundläggande dynamiskt system som Simulink vet hur det skall simuleras. Blocket representeras av insignaler, inre tillstånd och utsignaler, se figur 1



Figur 1: Simulink block, [5] s. 2-3

Blockets interna representation kan sammanfattas med ekvation 1.

$$\begin{aligned} y &= f_o(t,x,u) \\ x_{d_{k+1}} &= f_u(t,x,u) \\ x'_c &= f_d(t,x,u) \end{aligned} \qquad \text{där } x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

Ekvation 1: Intern blockrepresentation, [5] s. 2-5

f_0 är utsignalsfunktionen som beräknar blockets utsignal baserat på aktuell simuleringstid, interna tillstånd och insignaler. f_u kallas uppdateringsfunktion och används för att beräkna nästa diskreta tillstånd för de block som innehåller diskreta tillstånd. f_d beräknar derivatan av blockets kontinuerliga tillstånd. Blockets totala interna tillstånd kan bestå både av diskreta och kontinuerliga tillstånd.

En viktigt egenskap hos de flesta grundläggande block är möjligheten till parametrisering av viktiga funktioner. Detta gör att ett block egentligen representerar en familj av block som skiljer sig åt genom olika värden på parametrarna. Som parametrar till ett block kan man använda ett värde, en variabel eller ett uttryck. De flesta parametrar kan påverkas under simuleringens gång.

3.1.1.1 Tillstånd

För vissa block beror utsignalen på vilket tillstånd blocket för tillfället befinner sig i. Vilket nästa tillstånd blir beror i sin tur på tidigare tillstånd och nuvarande insignal. Integratorn är ett exempel på ett block som har ett inre tillstånd. Utsignalen är integralen av insignalen från simuleringens start till den nuvarande simuleringstiden och integralens värde från föregående beräkning sparas som ett internt tillstånd. Simulinks förstärknings block är däremot ett exempel på ett block som saknar inre tillstånd. Utsignalen är insignalen multiplicerat med förstärkningsfaktorn. Utsignalen bestäms därför endast av den nuvarande insignalen och en av blockets parametrar.

3.1.1.2 Kontinuerliga och diskreta block

Simulink innehåller både diskreta och kontinuerliga block. Kontinuerliga block svarar kontinuerligt på kontinuerliga insignaler medan diskreta block endast uppdateras vid tidpunkter som är heltals multipler av blockets sampelperiod. Diskreta block behåller samma utsignal mellan uppdateringarna. De diskreta blocken innehåller en parameter som specificerar blockets sampelperiod. Om ingen sampelperiod anges kan blocket ärva sampelperioden från de block som genererar blockets insignaler.

3.1.2 Subsystem

För att underlätta förståelsen av modellen är det möjligt att bygga en hierarki av subsystem. Genom att skapa en så kallad mask för sitt subsystem kan man låta systemet ha egna parametrar som anges genom en dialogruta. Till varje subsystemsparameter associeras en variabel som är lokal inom det aktuella subsystemet. Denna variabel kan sedan användas för att ange parametrar till de olika block som bygger upp subsystemet.

3.1.3 Lösare

Simulink simulerar det dynamiska systemet genom att beräkna modellens tillstånd och utsignaler vid ett antal tidpunkter under simuleringstiden. Det finns olika metoder för att beräkna modellens tillstånd och utsignaler och ingen metod är bra för alla typer av modeller. Processen att beräkna modellens tillstånd och utsignaler kallas att lösa modellen (eng. solv). För att lösa modellen tillhandahåller Simulink en mängd lösare (eng. solvers) för olika typer av modeller.

3.1.3.1 Fix-stegs lösare eller variabelstegs lösare

Simulinks lösare kan kategoriseras i två kategorier, fix-stegs lösare och variabelstegs lösare.

Fix-stegs lösare löser modellen med ett bestämt tidsintervall mellan de tidpunkter där beräkning sker. Intervallet mellan beräkningar kallas tidssteg (eng. time step). Längden på tidsteget kan specificeras av användaren eller väljas av lösaren. Generellt gäller att kortare tidssteg ger mer precision men tar längre tid att simulera.

Variabelstegs lösare varierar tidsteget genom att ta längre steg då modellens tillstånd ändras långsamt och minska tidsstegen då modellens tillstånd ändras snabbare. Beräkningen av tidsteget kräver resurser och tid under simuleringen men målet är att man ändå genom att ta färre steg kan minska simuleringstiden utan att förlora precision.

3.1.3.2 Kontinuerlig eller diskret lösare

Kontinuerliga lösare använder numeriska integrationsmetoder för att beräkna modellens kontinuerliga tillstånd. Det finns ett antal olika numeriska integrations metoder tillgängliga. Simulink använder så kallade ordinära differentialekvationslösare. Dessa metoder är de mest stabila, effektiva och precisa för numerisk integration, [5].

Diskreta lösare används framför allt för att lösa diskreta modeller. Lösarna beräknar modellens tillstånd varje tidssteg och inga beräkningar sker däremellan. Inga kontinuerliga tillstånd beräknas.

Det går att använda en kontinuerlig lösare men inte en diskret för en modell som innehåller både diskreta och kontinuerliga tillstånd.

3.2 Simulering

Simulering av ett dynamiskt system är processen för att beräkna en modells utsignaler och inre tillstånd, under en tid som bestäms av användaren. Beräkningarna baseras på den information som ingår i modellen och dess insignaler. Simuleringen i Simulink består av två faser, initieringsfasen och exekveringsfasen.

3.2.1 Initiering av simuleringen

Då simuleringen startas genomgår Simulink en procedur för att kontrollera att modellen inte innehåller några fel som gör att simuleringen inte kan startas. Vid initieringen allokeras också minne för simuleringens initialvärden, datatyper för modellens signaler kontrolleras och blocken sorteras för att rätt uppdateringsordning skall uppnås.

3.2.1.1 Sortering av block i uppdateringsordning

Vid simulering uppdaterar Simulink varje blocks utsignaler och interna tillstånd en gång varje tidsintervall. För att rätt resultat skall uppnås måste uppdateringarna av blocken ske i en bestämd ordning. Om till exempel ett blocks utsignal beror på dess insignal under det aktuella tidsintervallet måste blocket uppdateras efter det block som genererar insignalen för att utsignalen skall vara korrekt.

För att skapa en sortering av alla block som ingår i modellen som leder till ett korrekt resultat i varje tidsintervall karaktäriserar Simulink ett blocks insignaler efter relationen mellan insignalen och utsignalen för blocket. Om ett blocks utsignal beror på insignalen det aktuella tidsintervallet kategoriseras insignalen som direkt genomgående (eng. direct feedthrough). Ett exempel på block som har direkt genomgående insignal är förstärkningsblocket.

Vid initieringen genereras en uppdateringslista med alla block som inte har direkt genomgående insignaler först i listan utan bestämd inbördes ordning. Listan fullbordas med de block som har direkt genomgående insignaler i en ordning som garanterar att insignaler till efterföljande block alltid genereras före efterföljande block uppdateras.

För att Simulink skall kunna bestämma en ordning för hur blocken skall updateras får inga så kallade algebraiska loopar finnas i modellen. Ett exempel på en algebraisk loop (eng. algebraic loop) är om ett block med direkt genomgående insignal återkopplas via en loop som består av block som alla har direkt genomgående insignaler. Denna situation leder till att sorteringen hamnar i ett låst läge eftersom det är omöjligt att uppfylla den första regeln för uppdateringsordning. Denna situation måste därför undvikas när modellen konstrueras, [5].

3.2.2 Exekvering av simuleringen

Efter initieringen inleds exekveringsfasen av simuleringen. Under exekveringen beräknas modellens tillstånd och utsignaler vid tidpunkter som bestäms av simuleringens tidsteg. Vid varje tidpunkt genomförs följande steg i angiven ordning, [5] s. 2-10.

1 Uppdatera utsignaler för alla block i den ordning som anges av den sorterade listan för uppdateringsordning.

2 Uppdatera blockens inre tillstånd i den ordning som anges av listan.

3 Undersök modellens kontinuerliga tillstånd för att hitta diskontinuiteter.

4 Beräkna nästa tidsteg.

Utsignalerna beräknas med hjälp av blockens utsignalsfunktion. Diskreta blocks utsignal uppdateras endast vid heltalsmultipler av blockets sampelperiod. Blockens diskreta tillstånd beräknas med hjälp av blockets uppdateringsfunktion och kontinuerliga tillstånd beräknas genom att tidsderivatan av det kontinuerliga tillståndet beräknas. För att hitta diskontinuiteter i modellens kontinuerliga tillstånd använder Simulink en metod för att hitta nollgenomgångar (eng. zero-crossing detection).

Simulink repeterar steg 1-4 varje tidsteg tills slutet av simuleringstiden nås.

3.2.2.1 Diskontinuerliga tillstånd

Under simuleringen är det viktigt att undersöka modellens tillståndsvaribler för att hitta diskontinuiteter i dessa. Anledningen är att diskontinuiteter i tillståndsvariablerna ofta sammanfaller med viktiga händelser i simuleringen. Till exempel motsvaras en studs i en simulering av en studsande boll av en diskontinuitet i bollens hastighet.

Det är viktigt att området runt en diskontinuitet simuleras noga för att inte misstolkningar av resultatet skall göras. För att återgå till exemplet med den studsande bollen finns det en risk att tidpunkten för studsen faller mellan två tidssteg i simuleringen. Detta leder till att bollen ser ut att byta rikting utan att nå den yta den studsar mot.

Variabelstegs lösare kan vara lösningen på problemet genom att minska tidsstegen i närheten av en diskontinuitet då tillståndsvariablerna tenderar att ändras väldigt snabbt. Problemet med dessa är att tidstegen i närheten av en diskontinuitet riskerar att bli så små att simuleringstiden blir mycket lång. Simulink använder en teknik kallad nollgenomgångs detektering för att hitta diskontinuiteter i tillsåndsvariablerna utan att antalet tisdsteg blir för stort.

3.3 Modellering och simulering av tidsdiskreta system

Eftersom digital sekvensiell hårdvara är ett exempel på ett tidsdiskret system, där systemklockan används för att indikera när uppdateringar av systemets tillstånd skall ske har modellering och simulering av diskreta system varit en stor del av arbetet. Hybrida modeller har använts för att modellera analoga delar tillsammans med digitala.

Diskreta modeller i Simulink bygger på att vissa block har möjligheten att arbeta med diskret tidsskala genom sampelperiod parametern (eng. sample time). Dessutom finns möjligheten för ett block att ärva sampelperioden från det eller de block som genererar dess insignaler.

3.3.1 Sampelperiod

Genom att ange sampelperiod för ett block kan blocket fungera tidsdiskret. Sampelperioden kan specificeras antingen som ett värde eller som en vektor där det första värdet är sampelperioden och det andra är en offset. Kontinuerlig sampelperiod representeras av sampelperiod = 0.

Sampelperiod $= [T_s, T_o]$

Anger att uppdatering av blocket sker vid tidpunkter enligt

$$t_n = n * T_s + |T_o| \qquad n \in N$$

Ekvation 2: Sampelperiod för diskreta block

Det går inte att ändra sampelperioden för ett block under körning av simuleringen. Vid initieringen av simuleringen utreder Simulink sampelperioden för varje block . Ett blocks sampelperiod bestäms utifrån angiven sampelperiod (om användaren har angivit en explicit sampelperiod), genom att ärva sampelperiod från de block som genererar insignaler eller baserat på blockets typ (kontinuerliga block har alltid kontinuerlig sampelperiod). Det är denna sampelperiod som sedan används under hela simuleringen.

3.3.2 Lösare för tidsdiskreta system

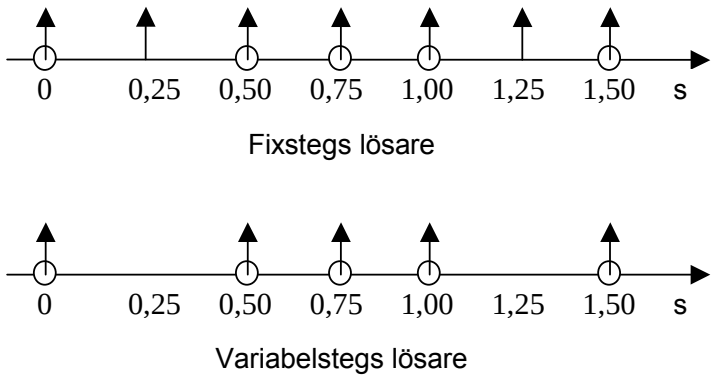
Fullständigt diskreta system kan simuleras med både diskreta och kontinuerliga lösare. Om en diskret lösare används löses modellen endast vid tidpunkter som är heltalsmultipler av den fundamentala sampelperioden för modellen. För mer information om fundamental sampelperiod se nedan

3.3.3 Beräkning av fundamental sampelperiod

Vid simulering av ett tidsdisket system med en fixstegs lösare måste modellen lösas vid varje tidpunkt som är en heltalsmultipel av den fundamentala sampelperioden för modellen. Detta är ett krav för att alla händelser i modellen skall kunna observeras.

Den fundamentala sampelperioden är den största gemensamma nämnaren för alla sampelperioder i modellen. Om till exempel två block har sampelperiod 0,25 s respektive 0,50 s är den fundamentala sampelperioden 0,25 s. Samma fundamentala sampelperiod har systemet även om blockens sampelperioder ändras till 0,50 s respektive 0,75 s.

Om man istället använder en variabelstegs lösare kan storleken på tidsstegen varieras så att beräkningar endast sker i de tidpunkter då blocken faktiskt uppdateras. Skillnaderna illustreras i figur 2 nedan. I figuren representerar pilarna tidstegen för lösaren och cirklarna representerar att blocken i modellen uppdateras.



Figur 2: Fix- resp variabelstegslösare, [5] s. 2-29

Om den fundamentala sampelperioden är mindre än någon av sampelperioderna för blocken i systemet kan man minska antalet tidssteg för lösaren om man använder en variabelstegs lösare. Om däremot den fundamentala sampelperioden sammanfaller med den kortaste sampelperioden i modellen är en fixstegs lösare att föredra eftersom ett optimalt antal steg då tas av lösaren utan att några resurser spenderas för att beräkna steglängden.

3.4 Mix-signal simulering

Analoga och mix-signal system simuleras traditionellt med SPICE liknande kretssimulatorer. SPICE använder mycket detaljerade modeller av komponenter vilket medför att simuleringarna går långsamt. Detta gör att simulering av systemkaraktärsistik som t ex bit-fels simuleringar i praktiken är omöjliga.

Genom att använda Simulink kan analoga och digitala komponenter simuleras i en och samma modell. Eftersom Simulink kan använda varibelstegslösare för att lösa ordinära differential ekvationer effektivt kan modellen simuleras effektivt och med hög precision, [6].

Exempel visar att Simulink kan användas för att effektivt modellera och simulera analoga och mix-signal system. Tester har gjorts av Motorola som använt Simulink istället för SPICE vid simulering av PLL funktioner. Dessa tester visar att simuleringstiden av en PLL kunde minskas från två timmar till under tio minuter. Precisionen i simuleringen kunde dock bibehållas på en tillräckligt hög nivå, [7].

4 FPGA design flöde

Detta avsnitt behandlar de steg som krävs för att generera en binär laddfil till en FPGA utifrån VHDL kod. I huvudsak behandlas det designflöde som stöds av de verktyg från Synopsys och Xilinx som använts i arbetet. VHDL kan också användas för att beskriva konstruktioner med annat slutresultat än FPGA t ex ASIC men här behandlas endast konstruktion med FPGA som slutmål.

4.1 FPGA

En FPGA eller Field Programmable Gate Array är en typ av programmerbar krets. De flesta FPGA:er på marknaden är så kallade SRAM baserade FPGA:er. Dessa FPGA:er kan konfigureras om inom några millisekunder och de kan rekonfigureras hur många gånger som helst. Dock tappar kretsen sin konfiguration då man slår av matningsspänningen, [8].

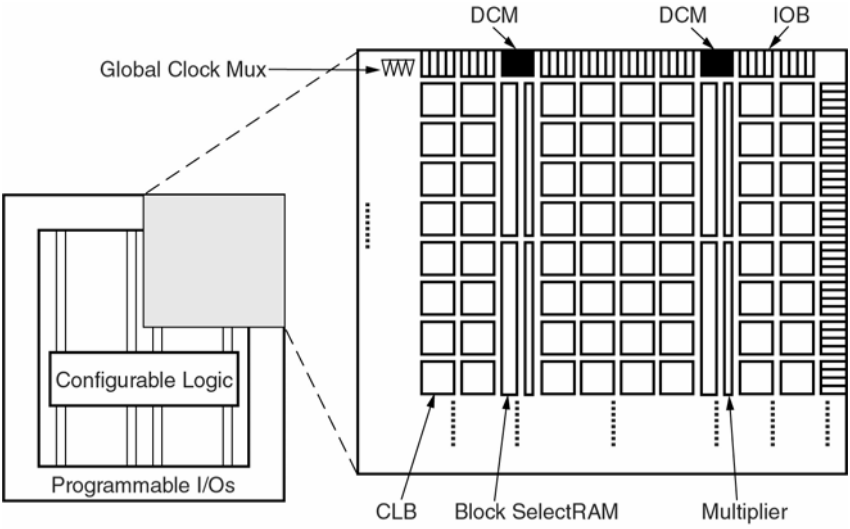
Under de senaste åren har utvecklingen inom FPGA området gått mycket snabbt framåt. För bara några år sedan motsvarade de största FPGA:erna tiotusentals grindar, klockfrekvensen för kretsarna låg kring 40 MHz och priset var ofta över 150$. Idag däremot kan en FPGA som motsvarar över en miljon grindar och arbetar med en klockfrekvens kring 300 MHz kosta kring 10$. Dagens största FPGA:er närmar sig 10 miljoner grindar och nya funktioner har också blivit tillgängliga såsom inbyggda processorer och minne, [9].

4.1.1 Xilinx Virtex II

Den FPGA som använts under arbetet är en Xilinx Virtex II XC2V2000-4 FG676. Kapaciteten hos kretsen motsvarar 2 miljoner grindar. Virtex II familjen från Xilinx innehåller elva olika kretsar med olika kapacitet upp till 8 miljoner grindar, [10].

4.1.1.1 Uppbyggnad

Xilinx Virtex II FPGA är uppbyggd av så kallade Configurable Logic Blocks (CLBs) tillsammans med specialiserade funktionsblock som t ex I/O block, klocknings kretsar och specialicerade multiplikatorer. Varje CLB är i sin tur uppbyggd av fyra slices. Slice:arna innehåller Look Up Tabeller (LUTs) som används för att realisera de logiska funktionerna. Digital Clock Managers (DCMs) används för att kompensera för interna fördröjningar i klockdistributionsnätet och kan också användas för att generera interna klocksignaler utifrån en extern klocka. I/O blocken är programmerbara för att kunna använda upp till 25 olika typer av off-chip signalering, figur 3.



Figur 3: Virtex II översikt [10]

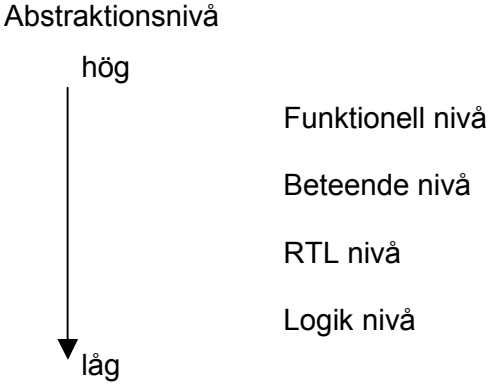
4.2 VHDL

VHDL står för Very high speed integrated circuit Hardware Description Language och är ett av de ledande språken för hårdvarubeskrivning. VHDL utvecklades under 80-talet med stöd från det amerikanska försvarsdepartementet. Den första standardiseringen av språket kom 1987. VHDL standardiserades för att beskriva och specificera ett system, inte för konstruktion, [11].

VHDL språket utvecklades för att beskriva hårdvara och först senare såg man möjligheterna att använda språket för konstruktion. Detta gör att endast en delmängd av språket är tillgängligt för konstruktion. Det innebär också att olika konstruktionverktyg ställer olika krav på hur den VHDL kod som används skall skrivas för att utnyttja verktyget optimalt. Det krävs därför att man som VHDL konstruktör har god kännedom om hur VHDL koden tolkas och implementeras av konstruktionsverkygen och dessutom anpassar sig till det verktyg man använder, [12].

4.2.1 Abstraktionsnivåer

VHDL innehåller möjlighet att använda flera olika abstraktionsnivåer, se figur 4. Den högsta abstraktionsnivån i VHDL är funktionell. På funktionell nivå beskrivs algoritmer utan information om hårdvaruimplementation eller tid. Beteende nivå är nästa abstraktionsnivå där även tid ingår i beskrivningen av algoritmen. Nästa mer detaljerade nivå är RTL nivån. RTL är en förkortning av Register Transfer Level och denna nivå består av ett språk som innehåller komponenter som har motsvarigheter i hårdvara som t ex adderare, minnen och register. På RTL nivå är samtliga register i den slutliga realiseringen definerade i VHDL koden. Den lägsta abstraktionsnivå som stöds av VHDL är logik- eller grindnivå. Det är en beskrivning av systemet med boolesk algebra eller ett grindnät.



Figur 4: Olika abstraktionsnivåer i VHDL

4.2.1.1 Strukturell VHDL

Strukturell VHDL är en nätlista som innehåller komponeter och deras sammankopplingar. Det är en hierarkisk konstruktion där varje komponent i sin tur kan innehålla en hierarki av subkomponenter. Strukturell VHDL kan göras grafiskt som ett blockshema med hjälp av olika verktyg. Innehållet i varje block kan sedan specificeras som ytterligare blockdiagram eller som VHDL kod som beskriver funktionen.

4.2.2 Syntes

Med syntes av VHDL menar man processen att överföra en beskrivning från en högre abstraktionsnivå till en lägre. Detta kan t ex vara en transformation av koden från beteende- till RTL nivå eller en övergång från en beteende beskrivning till en strukturell beskrivning.

4.2.2.1 RTL syntes

RTL syntes innebär att VHDL kod skriven på RTL nivå används för att generera en nätlista som innehåller komponenter och deras sammankopplingar. Nätlistan kan vara representerad i olika filformat, EDIF (Electronic Design Interchange Format) är vanligt men t ex Xilinx använder ett eget format som heter NGC.

De flesta avancerade syntesverktyg genererar en generell nätlista utifrån VHDL koden. Detta möjliggör optimering av konstruktionen. När verktyget översätter VHDL koden till en generell nätlista identifieras vissa specifika konstruktioner såsom t ex register, adderare och multiplicerare. Dessa funktioner kan implementeras på flera på förhand kända sätt t ex kan en adderare implementeras med en ripple-carry adderare eller en carry look ahead adderare. Dessa båda skiljer sig åt när det gäller hastighet och area. Baserat på de krav användaren ställer på maximal area och fördröjning kommer verktyget välja den typ av adderare som tar upp minst area men ändå uppfyller tidskraven. RTL syntes optimerar inom varje klockcykel för att uppfylla tisdkraven.

Exempel på verktyg för RTL syntes är XST från Xilinx och Synplify från Synopsys.

4.3 FPGA implementation

FPGA implementationen innebär att den nätlista som genererats vid syntesen av VHDL koden översätts till en laddfil för en specifik FPGA. Eftersom olika typer av FPGA:er har olika egenskaper är resultatet och även de delresultat som erhålls vid implementationsprocessen beroende på vilken FPGA som används.

Detta avsnitt använder Xilinx FPGA:er och utvecklingsverktyg som utgångspunkt och variationer kan därför förekomma om andra kretsar eller vektyg används.

4.3.1 Nätlista

Implementations processen inleds med att en nätlista skapas utifrån VHDL koden. Detta är detsamma som RTL syntes men beroende på vilket syntesverktyg man använder kan man redan på denna nivå optimera nätlistan för att passa den krets man har tänkt använda. Utdata skrivs i en EDIF eller NGC fil.

För att uppnå det önskade resultatet kan användaren specificera vissa krav (eng. constraints) som implementationen måste uppfylla. Dessa krav kan vara tidskrav och areakrav men också en mappning av I/O signalerna i VHDL koden till specifika kontakter på FPGA:n. Under skapandet av nätlistan undersöks dessa krav för att göra en preliminär bedömning av hur väl det slutgiltiga resultatet uppfyller kraven.

I mitt arbete har jag använt Synplify från Synopsys för att skapa en nätlista. Nätlistan har optimerats för Xilinx XC2V2000-4 FPGA. Den resulterande nätlistan genereras som en fil i EDIF format. De krav som finns på kretsen sparas i en NCF fil som innehåller de tidskrav som ställs på det slutgiltiga resultatet.

4.3.1.1 NGDBuild

Xilinx använder ett filformat som kallas NGD (Native Generic Database) för att representera nätlistan. NGD filen innehåller både den generiska representationen av nätlistan och en Xilinx specifik nätlista där alla komponenter kommer från Xilinx bibliotek med realiserbara komponeter. Det är den Xilinx specifika nätlistan som används för att mappa nätlistan till en FPGA komponent, (mer om mappning nedan). NGD filen innehåller dessutom de krav som användaren specificerat.

NGDBuild är det program Xilinx utvecklingsverktyg använder för att skapa en nätlista i NGD format. Indata till NGDBuild är resultatet av RTL syntesen, dvs en EDIF eller NGC fil beroende på vilket verktyg som använts för att skapa nätlistan.

4.3.2 Mappning

Mappning innebär att den logiska beskrivningen av konstruktionen överförs till en FPGA specifik representation där alla logiska komponenter motsvaras av komponenter i den FPGA som används. De logiska funktioner som används har en motsvarande realisering i en CLB eller I/O resurser i FPGA:n.

Vid mappningen undersöks också nätlistan för att hitta eventuell överflödig logik som då tas bort. Utdata från mappningen är en NGD fil som innehåller en beskrivning av konstruktionen i termer av komponeter i FPGA:n. Dessutom genereras filer som innehåller krav på den fysiska implementationen. Dessa krav är en översättning av de krav som användaren specificerar på till exempel vilka I/O signaler som använder vilken kontakt på FPGA:n.

4.3.3 Place and Route (PAR)

Vid place and route görs den slutgiltiga implementationen i FPGA:n. Place innebär att de mappade komponenterna tilldelas en specifik fysisk resurs i FPGA:n. Det innebär t ex att en specifik CLB tilldelas en viss logisk funktion eller en viss multiplikation tilldelas en bestämd multiplikator. Olika fysiska krav som specificerats vid mappningen används för att styra place funktionen. Routing innebär att komponenterna kopplas samman via kontaktnätet i FPGA:n.

PAR processen använder de krav som användaren angivit för att optimera arbetet så att kraven uppfylls. En statisk timing analys genomförs för att uppskatta fördröjningarna i kretsen. Det är först efter routing ledningsfördröjningar kommer med i analysen av fördröjningarna i kretsen.

I PAR rapporten som genereras av verktyget kan man få information om hur väl implementationen uppfyller de krav användaren ställt upp. Här får man också information om maximal klockfrekvens för FPGA:ns systemklocka m m.

4.3.3.1 PAR optimering

PAR verktyget genomför en optimeringsprocess för att hitta en realisering som uppfyller alla krav samtidigt som den utnyttjar så lite som möjligt av de resurser som finns tillgängliga i FPGA:n. Vid denna optimering görs en avvägning mellan graden av optimering och den tid det tar att genomföra PAR.

Om de krav användaren ställt på den slutliga implementationen inte kan uppfyllas framgår det om inte tidigare efter place and route. Exempel på krav som inte kan uppfyllas är t ex tidskrav om en logikkedja mellan två register innebär för lång fördröjning för den önskade klocktakten. För att lösa denna typ av problem kan man använda två metoder. Genom att ändra inställningarna till PAR verktyget kan man öka optimeringen i hopp om att en bättre realisation kan hittas. Alternativ två är att gå tillbaka till VHDL beskrivningen och introducera ett eller flera pipeline steg för att förkorta den kritiska fördröjningen.

4.3.4 Generering av laddfil

Utdata från PAR är en NGD fil som innehåller en fullständig beskrivning av implementationen. Denna fil kan sedan användas för att generera en binär laddfil till FPGA:n som innehåller information för att konfigurera FPGA:n.

5 Xilinx System Generator for DSP

Xilinx System Generator for DSP (Digital Signal Processing) är ett verktyg för att modellera och konstruera FPGA-baserade digitala signalbehandlingsfunktioner i Simulink, [13]. Verktyget erbjuder en hög abstraktionsnivå och ett grafiskt användargränssnitt samtidigt som en direkt realisering i en FPGA är möjlig.

System Generator är ett försök från Xilinx att göra FPGA:er mer tillgängliga för användare av digitala signalprocessorer. Genom att använda System Generator kan DSP ingenjören använda MATLAB/Simulink för att utveckla algoritmer som sedan enkelt kan realiseras i en FPGA, [14].

Xilinx System Generator består av ett blockbibliotek till Simulink, det så kallade Xilinx blockset, samt mjukvara för att generera en VHDL representation av Simulinkmodellen. I Simulink kan block från Xilinx blockset kombineras med block från andra blockbibliotek för simulering. System Generator kan däremot endast realisera block från Xilinx blockset i hårdvara.

5.1 Modellering

Simulink stöder modellering av tidsdiskreta system vilket gör det möjligt att modellera digital hårdvara. System Generator tillåter dessutom att modellen används för att skapa en realisering av den digitala funktionen i en FPGA.

Modellering med Xilinx blockset skiljer sig på några punkter från modellering med övriga Simulink block. Ett exempel på skillnader är de olika datatyper som används för att representera signaler i modellen.

Många block i Xilinx blockset har parametrar som påverkar den resulterande hårdvaran. För att uppnå en högpresterande realisation av modellen kräver modellering med Xilinx blockset viss erfarenhet av konstruktion av digitala funktioner i allmänhet och FPGA konstruktion i synnerhet. Exempelvis måste konstruktören ha erfarenhet av att arbeta med fixtalsrepresentation av data.

5.1.1 Representation av data

Xilinx System Generator stöder tre olika datatyper, flyttal med dubbel precision, tvåkomplement representerade fixtal och positiva fixtal. Flyttal kan inte användas i modeller som skall realiseras i hårdvara utan är endast tillgängliga för simulering.

Genom att flyttal stöds för simulering är det möjligt att introducera fixtals aritmetik succesivt och därmed kan kvantiserings beteende behandlas separat från den matematiska algoritmen. I praktiken innebär det att en matematisk modell först kan utvecklas och när den visat sig fungera med flyttals representation kan kvantisering införas.

För att möjliggöra utbyte av data mellan block från Xilinx blockset och övriga Simulink block som alltid använder flyttal för att representera data används så kallade Gateway block, figur 5.

Figur 5: Xilinx blockset gateway block

5.1.2 Avrundning och overflow

En fördel med möjligheten att införa kvantisering succesivt är att konstruktören enkelt kan undersöka behovet av mättnadslogik respektive trunkering och avrundning.

Det finns block i Xilinx blockset för att ändra fixtalsrepresentationen av data. Det är möjligt att påverka om mättnadslogik skall användas vid overflow. Vid trunkering av de minst signifikanta bitarna kan avrundning användas.

Xilinx blockset innehåller dessutom en funktion för att låta Simulink beräkna hur data skall representeras för att overflow skall undvikas. Genom att använda denna funktion kan en första fixtalsrepresentation erhållas som sedan kan optimeras av konstruktören.

5.1.3 Klockning

I Simulinkmodellen finns inga explicita klocksignaler. Detta gör att man heller inte har tillgång till några klocksignaler vid konstruktionen av sin modell. Det är istället modellens fundamentala sampelperiod som motsvarar den klocka som den resulterande FPGA realiseringen använder, (se avsnitt 3.3.3 för mer information om fundamental sampelperiod).

5.1.3.1 Flera Klockdomäner

Olika klockdomäner i modellen representeras av olika datatakt för modellens signaler. För att modellera olika klockdomäner kan man använda de upp- och nedsamplingsblock som ingår i Xilinx blockset. Simulink innehåller dessutom en funktion för att färgkoda de olika delar av modellen som använder olika sampelperioder. Det går endast att göra modeller där de olika klockorna har en period som kan skapas genom att kombinera upp- och nedsamplingar av den fundamentala klockperioden.

De olika klockdomänerna realiseras med hjälp av clock enable (CE) signaler (se avsnitt 5.2.3.1). Dessa CE signaler kan användas i modelleringen för att underlätta synkronisering mellan olika klockdomäner. För att få tillgång till CE signalerna vid modellering används ett Clock Enable Probe block.

5.1.3.2 Hardware oversampling

Vissa block i Xilinx blockset har en parameter som möjliggör så kallad hardware oversampling. Hardware oversampling innebär att den interna datatakten i blocket är högre än sampeltakten för in- och utsignalerna. Ett exempel på ett block som använder hardware oversampling är Xilinx blocksets FIR filter block. Genom att använda en högre klocktakt internt kan filtret realiseras med en mer seriell struktur och därmed kan en mer areaeffektiv realisering uppnås.

5.1.4 Synkronisering

Några av blocken i Xilinx blockset innehåller funktioner för att underlätta synkronisering av dataflöde genom modellen. Valid in och valid out signaler används för att indikera när ett block kan ta emot data respektive när resultatet av en beräkning finns tillgängligt.

5.1.5 Modellering med VHDL

Det är möjligt att använda VHDL för att göra modeller av delsystem och använda dessa VHDL modeller i Simulink. Ett speciellt block från Xilinx blockset, en så kallad svart låda (eng. black box), används i Simulinkmodellen för att möjliggöra VHDL modeller i Simulink.

Vid simulering i Simulink finns två alternativ för hur VHDL modellen simuleras. Det går att använda VHDL modellen enbart för realisering och använda en Simulinkmodell som innehåller block från andra block än Xilinx blockset vid simuleringen. Denna metod ger kortast simuleringstid men kräver att en simuleringsmodell uppbyggd av Simulink block skapas för VHDL koden. Alternativet är att samsimulera Simulinkmodellen med VHDL koden via ModelSim som är en VHDL simulator från Modeltech, Inc. Denna metod medför längre simuleringstid eftersom kommunikation mellan Simulink och ModelSim måste genomföras.

5.2 Realisering

Med System Generator är det möjligt att generera en FPGA realisering av sin modell. Hårdvarurealisationen genereras med hjälp av mjukvara som översätter blocken i Xilinx blockset till en representation i VHDL som går att syntetisera och implementera i en Xilinx FPGA. Den VHDL kod som genereras är optimerad för användning med Xilinx FPGA familjer.

5.2.1 VHDL generering

Det är enkelt och går fort att generera VHDL kod från en fungerande modell. Xilinx erbjuder bra support för eventuella problem som kan uppstå vid generering av kod.

Vid generering av VHDL specificerar användaren vilken FPGA familj, kretstyp, och vilket syntesverktyg som skall användas vid implementationen. System Generator genererar dessutom filer för att underlätta implementationen. T ex genereras projektfiler för Xilinx implementations verktyg och filer som innehåller de tidskrav som måste uppfyllas för att korrekt funktion skall uppnås.

Vid genereringen använder System Generator de gateway block som ingår i modellen för att översätta dem till in- och utsignalen i FPGA realisationen. Genom att ange parametrar till gateway in blocken kan användaren styra hur insignalerna till VHDL koden ser ut, t ex hur många bitar signalen använder.

5.2.1.1 Testbänks generering

Vid generering av VHDL har användaren möjlighet att dessutom generera en testbänk som kan användas för att verifiera funktionen hos den kod som genererats. Verifieringen görs med hjälp av en VHDL simulator. Testbänken innehåller insignals stimuli som motsvarar insignalerna till de gateway in block som finns i modellen. Dessutom kontrollerar testbänken data på utgångarna från VHDL koden och jämför denna med utsignalerna från gateway out blocken i modellen.

System Generator genererar dessutom makro filer för att enkelt kunna göra verifiering av VHDL koden med hjälp av ModelSim.

5.2.1.2 Generering av krav

Bland de filer som genereras finns de krav som måste uppfyllas av implementationen för att korrekt funktion skall uppnås. Exempel på de krav System Generator genererar är krav för systemklockans period. Dessutom kan användaren specificera hur in- och utsignalerna i modellen skall mappas till kontakter på FPGA:n. Dessa krav inkluderas också bland de krav som System Generator specificerar för syntesverktygen.

5.2.2 Användning av IP kärnor

För att uppnå en effektiv realisering av modellen utnyttjar Xilinx System Generator så kallade intellectual property (IP) kärnor (eng. IP core). Dessa kärnor är optimerade för att utnyttja FPGA:ns resurser och uppnå maximal prestanda. IP blocken genereras med hjälp av ett verktyg som heter Xilinx Core Generator. Vid genereringen anropar System Generator Core Generator med de parametrar som användaren angivit för de olika blocken för att generera en anpassad IP kärna.

System Generator kan använda IP kärnor för många block i Xilinx blockset. Användaren kan själv välja vilka block som skall implementeras med IP kärnor och vilka som skall representeras av syntetiserbar VHDL. Vissa block kan endast realiseras som IP kärnor. FFT blocket är ett sådant exempel. Många enklare block har ingen IP motsvariget utan representeras alltid med syntetiserbar VHDL.

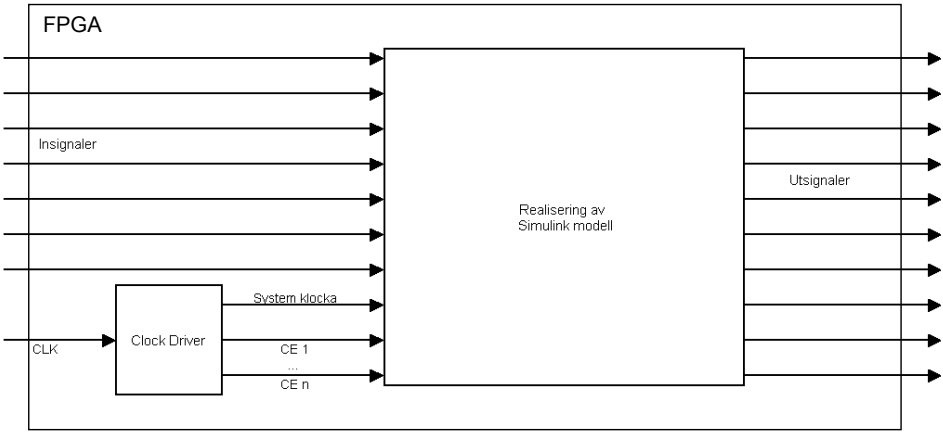
IP blocken som används är verifierade av Xilinx vilket garanterar korrekt intern funktion. De flesta IP block som används av System Generator kräver ingen ytterligare licens för att användas. Det finns dock stöd i System Generator för att använda vissa IP block från Xilinx som kräver ytterligare licenser. Dessa block indikeras i Xilinx blockset genom att färgen på blocken är grön istället för blå.

5.2.3 Realisering av systemklockan

Vid modellering är det viktigt att känna till kopplingen mellan Simulinks sampelperioder och systemklockan i FPGA realiseringen. Vid genereringen kan användaren specificera förhållandet mellan modellens fundamentala sampelperiod och perioden för systemklockan. I den genererade VHDL koden skapas en klockingång för systemklockan. Det är inte möjligt att generera mer än en klockingång.

5.2.3.1 Flera klockdomäner

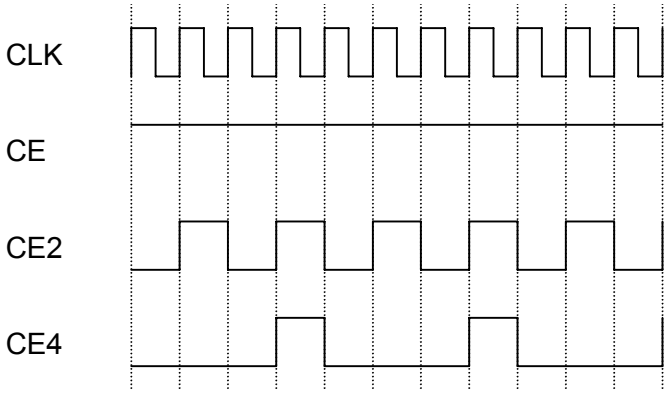
För att realisera en modell som innehåller flera klockdomäner använder System Generator clock enable (CE) signaler. Realiseringen sker genom att en komponent skapas i VHDL koden som genererar CE signalerna. Den komponent som används för att generera CE signalerna kallas Clock Driver i VHDL koden. Denna komponent tillkommer vid genereringen av VHDL och finns inte tillgänglig i Simulinkmodellen, se figur 6.



Figur 6: Principskiss för FPGA realisering av CE generering

CE signalerna används för att aktivera ett block under en klockcykel, blocket deaktiveras sedan under ett antal klockcykler. För att realisera en klockdomän som använder fyra gånger längre sampelperiod används en CE signal som är låg under tre klockcykler och sedan hög under en. Detta mönster upprepas sedan för att aktivera blocket var fjärde klockcykel. Denna realisering innebär att en enda systemklocka används i hela FPGA:n.

För varje sampelperiod i modellen finns en motsvarande CE signal. Om ett block har en sampelperiod som är fyra gånger längre än den fundamentala sampelperioden skapas en CE4 signal som aktiverar blocket var fjärde klockcykel. Figur 7 visar hur CE signalerna förhåller sig till systemklockan.



Figur 7: Generering av CE signaler för multipla klockdomäner

Det genereras en CE signal också för den del av kretsen som använder den fundamentala sampelperioden. Denna signal, CE, är hög under alla klockcykler.

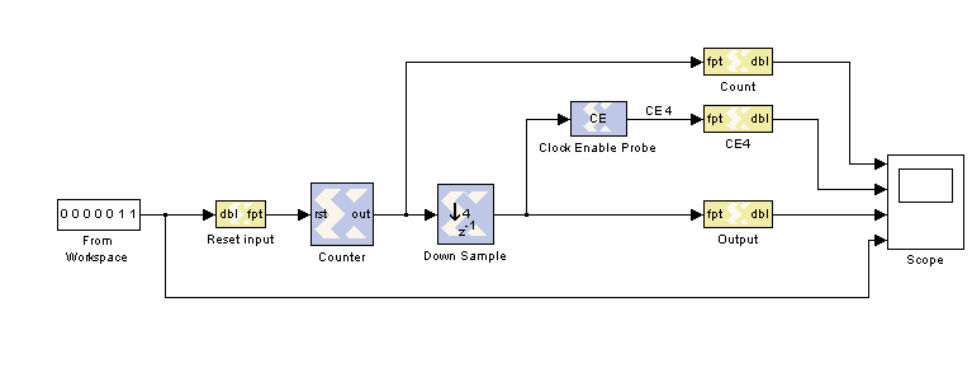
Om en modell innehåller flera klockdomäner genereras förutom krav på systemklockans period också tidskrav på de delar av systemet som använder längre klock perioder. Dessa krav inkluderas i den fil som innehåller övriga krav på realiseringen.

5.2.3.2 Synkronisering av klockdomäner

Vid användning av flera interna klockdomäner är synkronisering mellan de olika klockdomänerna viktigt. Synkronisering krävs för att data skall kunna utbytas mellan olika klockdomäner på ett säkert sätt. Det finns metoder för att överföra data mellan två asynkrona klockdomäner, t ex genom handskakning, men då dessa inte används av System Generator är de inte av intresse här.

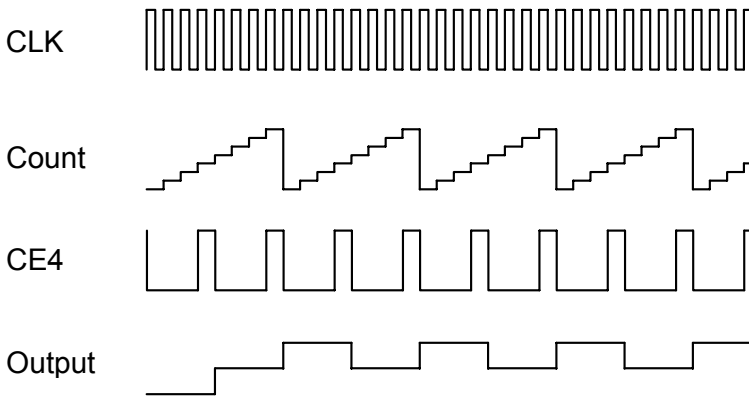
5.2.3.3 Decimering, ett exempel

Ett enkelt exempel på dataöverföring mellan två klockdomäner är nedsampling. Följande exempel visar hur nedsampling för att välja ut ett av fyra sampel går till med hjälp av Xilinx blockset. Exemplet används dessutom för att illustrera ett problem med den strategi som System Generator använder för att realisera olika klockdomäner.



Figur 8: Nedsampling med Xilinx blockset

Modellen i figur 8 använder ett nedsamplingblock för att sampla räknaren. Decimeringsfaktorn 4 innebär räknarens utsignal samplas vid den fjärde, åttonde, tolfte osv stigande klockflanken av systemklockan. Nedsamplingsblocket realiseras med ett register som använder CE4 signalen för att aktiveras var fjärde klockperiod.



Figur 9: Resultat av simulering av modellen

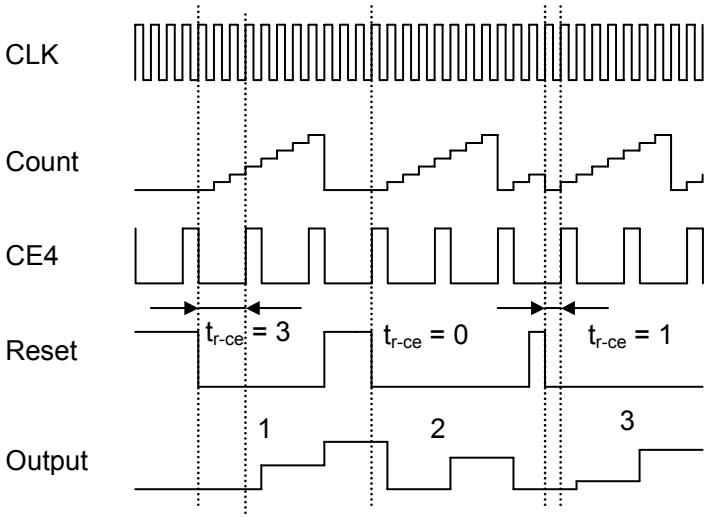
I figur 9 visas hur resultatet av nedsamplingen ser ut. Vid simuleringens start är registret nollställt. Nedsamplings registret läser in ett nytt värde vid CE4 signalens stigande flank. På grund av fördröjningen med en klockcykel genom registret presenteras resultatet på Output signalen en klockcykel senare. Ovan har en trebitars räknare använts för att räkna mellan 0 och 7. Nedsamplingsblocket väljer ut sampel nummer 4 och 8 från räknaren dvs värde 3 och 7.

Exemplet ovan visar hur Xilinx blocksets nedsamplingsblock kan fungera. Problem uppstår dock om reset signalen används i modellen ovan.

5.2.3.4 Problem med reset av flera klockdomäner

Reset signaler används i alla typer av sekvensiell logik för att återställa kretsen till ett känt utgångsläge. Reset signalen återställer alla minneselement i kretsen och säkerställer att kretsen alltid ger samma resultat vid samma sekvens av insignaler. Reset signalen skall dessutom vara oberoende av kretsen den används för att styra. Detta innebär att det t ex inte går att använda en utsignal från kretsen för att styra när en reset är möjlig. Detta krav kommer från det faktum att FPGA:n ofta är en del i ett större system där flera kretskort ingår och en systemreset måste kunna genomföras.

Reset signalen till räknaren i exemplet ovan används för att nollställa räknaren. För att Simulink skall acceptera reset signalen vid simuleringen måste sampelperioden för reset signalen vara samma som sampelperioden för räknaren. Det är räknarens sampelperiod som är den fundamentala sampelperioden för modellen.



Figur 10: Reset av räknare i decimerings exempel

I figur 10 illustreras hur reset signalen påverkar utsignalen av nedsamplingen. Fall 1 känns igen från figur 9. Räknaren samplas 4 respektive 8 klockcykler efter resetsignalens fallande flank och resultatet känns igen från figur 9.

Fall 2 och 3 visar på ett problem med det system med CE signaler som System Generator använder. I fall 2 samplas räknaren vid första, femte och nionde stigande flanken efter reset signalens fallande flank vilket ger utsignalen 0,4,0. I fall 3 samplas räknaren vid andra och sjätte stigande flanken efter reset signalens fallande flank. I detta fall fås utsignalen 1,5.

Det är tydligt att resultatet av nedsamplingen beror på tiden mellan resetsignalens fallande flank och CE4 signalens nästa stigande flank. Denna tid har indikerats $t_{r\text{-}ce}$ i figur 10. $t_{r\text{-}ce}$ mäts här i klockcykler för systemklockan. Ytterligare ett fall med $t_{r\text{-}ce} = 2$ kan uppstå men detta har inte tagits med i figuren.

I exemplet ovan har den viktigaste funktion med reset signalen, att återställa kretsen till ett känt utgångsläge satts ur spel. Beroende på hur lång tid som förflutit mellan systemstart och resetsignalens fallande flank kommer kretsen hamna i ett av fyra tillstånd. Om systemet dessutom innehåller ytterligare klockdomäner inses att antalet möjliga tillstånd kretsen kan hamna i efter en reset ökar. Detta beteende är inte acceptabelt.

5.2.3.5 Lösning av problemet

Ett förslag på lösning är att den CE signal med längst period används för att synkronisera reset signalen. Genom att läsa av CE signalen kan användaren se till att reset signalen alltid faller samtidigt som CE signalen. Detta är dock inte ett användbart alternativ eftersom reset signalen skall vara oberoende av den krets den används för att återställa.

Xilinx support har kontaktats för att försöka hitta en lösning på problemet. Utvecklarna av System Generator känner till problemet och en lösning är under utveckling. I väntan på att en tillfredställande lösning skall arbetas fram har en temporär lösning använts.

För att lösa problemet temporärt har den genererande VHDL koden modifierats. Modifieringen innebär att reset signalen som används för att återställa realiseringen av modellen dessutom kopplas till clock driver komponenten (se figur 6). För att möjliggöra reset av clock driver komponenten var en modifiering av VHDL koden som representerar komponenten nödvändig.

Genom att clock driver komponenten också nollställs startar även genereringen av CE signaler om vid en reset och fallet med $t_{r\text{-}ce} = 3$ från figur 10 kan garanteras.

Denna lösning innebär att realiseringen av modellen inte längre har identiskt beteende med modellen. Detta gör att en av de viktigaste fördelarna med System Generator går förlorad. Denna lösning måste därför ses som en temporär lösning i väntan på att en permanent lösning kan tas fram.

6 Xtreme DSP utvecklingskort

För att utvärdera Xilinx System Generator har ett utvecklingskort använts. Utvecklingskortet ingår i Xilinx Xtreme DSP Development Kit och tillverkas av Nallatech, Inc. Denna beskrivning behandlar främst de delar av plattformen som använts under arbetet. Xtreme DSP plattformen innehåller ytterligare funktionalitet som inte diskuteras här.

Figur 11: Xtreme DSP utvecklingskort

Kortet är en plattform anpassad för avancerade DSP funktioner. Bland funktionerna ingår dubbla A/D-omvandlare och dubbla D/A-omvandlare, SSRAM minne samt en Xilinx Virtex II FPGA. I paketet ingår dessutom mjukvara för att programmera FPGA:n via USB kommunikation med PC, [15].

Plattformen Xtreme DSP är uppbyggd av två moduler. Ett moderkort BenONE och en tilläggsmodul BenADDA.

6.1 BenONE

BenONE innehåller hårdvara för kraftförsörjning av plattformen, kommunikation med PC via PCI och olika klocknings alternativ för plattformen. Dessutom finns en USB modul för kommunikation med PC.

BenONE innehåller inga FPGA resurser som är tillgängliga för användaren.

6.2 BenADDA

BenADDA är en tilläggsmodul som innehåller två A/D-omvandlare, två D/A-omvandlare, SSRAM minne samt en Xilinx Virtex-II XC2V2000-4FG676 FPGA som är tillgänglig för användaren. Ytterligare funktioner inkluderar dioder som är anslutna till FPGA:n och möjlighet att klocka FPGA:n med en extern klocksignal.

6.2.1 A/D-omvandlare

De A/D-omvandlare som ingår i BenADDA modulen är två AD6644 från Analog Devices. AD6644 använder 14 bitar och 65 MHz samplingsfrekvens. De digitala utsignalerna använder 3,3 V CMOS kompatibel utnivå och två komplement representation, [16].

6.2.2 D/A-omvandlare

BenADDA innehåller två D/A-omvandlare av typen AD9772A från Analog Devices. Den maximala insignalsfrekvensen till omvandlarna är 160 MHz. Upplösningen är 14 bitar. D/A-omvandlarna innehåller dessutom interpolationsfilter med interpolationsfaktor 2 och möjlighet till nollinskjutning. D/A-omvandlarna använder binär offset representation för den digitala insignalen, [17].

6.3 Klockning av FPGA

Xtreme DSP plattformen erbjuder flera alternativ för klockning av den FPGA användaren har tillgång till.

BenONE innehåller två olika programmerbara klocksignaler, en oscillator och dessutom möjlighet att via PCI förse FPGA:n med klocksignal. Dessutom erbjuder BenADDA en extern klockingång och ytterligare en oscillator, [15].

6.3.1 Klocknings FPGA

För att administrera klocksignaler till de olika delarna av BenADDA används en speciell klocknings FPGA. Klocknings FPGA:n kan programmeras att använda klocksignaler från BenONE eller extern klocka. Klocknings FPGA:n används dessutom för att förse A/D- och D/A-omvandlarna med klocksignaler.

6.4 Programmering av FPGA

FPGA:n på BenADDA kan programmeras genom ett grafiskt gränssnitt som ingår i utvecklingsmiljön för Xtreme DSP plattformen. Det finns dessutom stöd för andra programmeringsmetoder t ex JTAG, [15].

7 Modulator

För att testa Xilinx System Generator och möjligheterna att använda Simulink för att generera en FPGA implementation har en modulator för en radiolänk konstruerats. Modulatorn har realiserats med hjälp av Xtreme DSP utvärderingskortet.

7.1 Bakgrund

För att uppskatta hur bra den VHDL kod som Xilinx System Generator genererar är behöver man jämföra den slutliga FPGA realisationen med en motsvarande realisation som utgår från handskriven VHDL kod. Det har också varit viktigt att testa Xilinx System Generator i ett testfall som liknar de konstruktioner som görs på Ericsson Microwave Systems.

Valet av modulatorn som testfall motiveras av att en liknande konstruktion tidigare tagits fram och implementerats i en FPGA. Därmed är det möjligt att direkt jämföra resultatet från Xilinx System Generator med den befintliga konstruktionen. Komplexiteten motsvarar också de konstruktioner som verkligen byggs i FPGA:er på EMW.

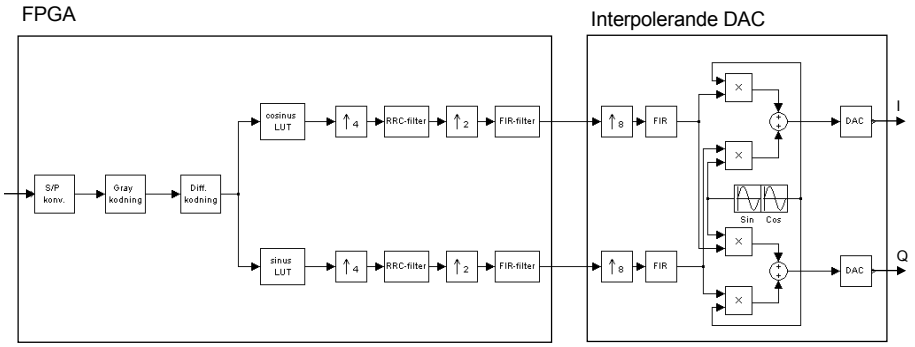
Målet med arbetet har varit att realisera en FPGA funktion och jämföra denna med motsvarande FPGA realisation som gjorts utifrån handskriven VHDL kod. Samt att se om ett utvärderingskort skulle kunna användas för att göra en prototyp av ett liknande system i framtiden.

7.2 Beskrivning av modulatorn

För att inte för mycket tid skall gå åt till själva konstruktionsarbetet har en befintlig modulator varit utgångspunkten för konstruktionen och endast mindre förändringar har gjorts för att passa in på Xtreme DSP kortet.

7.2.1 Beskrivning av befintlig modulator

Enkelt kan modulator funktionen beskrivas med att en seriell indataström moduleras med en av två möjliga metoder beroende på vilken mod systemet arbetar i. De två olika moderna möjliggör överföringshastigheter på 4 resp. 8 Mbit/s. Modulatorn innehåller också en filtrering av signalen med ett root raised cosine filter för att minska bandbredden för signalen, samt en digital uppblandning till IF. Detta genomförs i den befintliga konstruktionen på ett kort som innehåller en FPGA som sköter modulationen och root raised cosine filtreringen, en interpolerande D/A omvandlare som gör uppblandningen till IF samt en analog uppblandning till RF. Ett principiellt blockschema för den digitala modulatorfunktionen visas i figur 12.

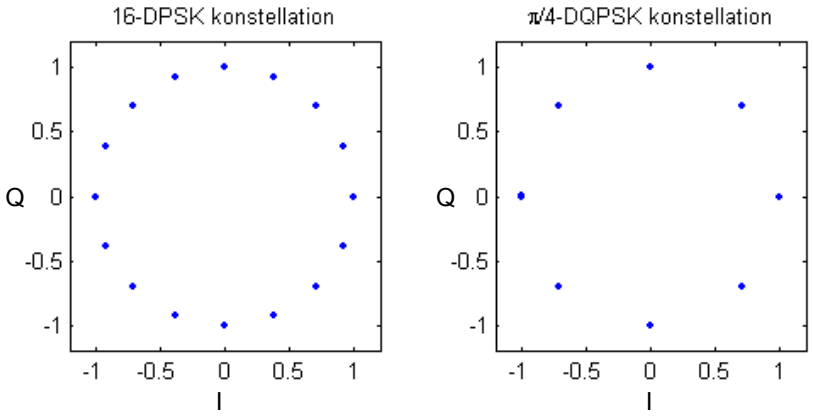


Figur 12: Principiellt blockshema för digital modulatorfunktion

7.2.1.1 FPGA funktion

FPGA:n kan arbeta i två olika moder beroende på vilken överföringshastighet som används. Den ena moden använder en π/4-DQPSK modulering och den andra moden använder en 16-DPSK modulering. Överföringshastigheten är 4 resp 8 Mbit/s. Figur 13 illustrerar de olika modulations konstellationerna.

Samplingstakten för insignalen till modulatorn är 4,608 MHz resp. 9,216 MHz beroende på vilken mod som används. I 4 Mbit/s mod består en symbol av två bitar medan en symbol består av fyra bitar i 8 Mbit/s moden. Detta ger en symboltakt på 2,304 MHz i båda moderna. Symbolerna gray- och differential kodas och med hjälp av en tabell mappas symbolerna till rätt I- och Q-värde.



Figur 13: Modulations konstellationer

Efter moduleringen sker en interpolering med en faktor 4 och I- och Q-signalerna filtreras med ett root raised cosine filter för att minska bandbredden. Efter root raised cosine filtreringen sker ytterligare en interpolering denna gång med en faktor 2. Ett interpoleringsfilter används för att filtrera bort oönskade frekvenskomponenter som uppstår vid interpolationen. Detta ger en samplingstakt ut från FPGA:n som är 18,432 MHz.

7.2.1.2 Interpolerande D/A-omvandlare

D/A-omvandlaren innehåller två kanaler en för I- och en för Q-signalen. D/A omvandlaren ökar signalernas datatakt 8 gånger. Detta sker genom tre stycken interpoleringar var och en med en faktor 2. Varje interpolering följs av ett interpoleringsfilter. D/A-omvandlaren innehåller dessutom en I/Q mixer som används för att blanda upp signalen till 36,864 MHz IF.

I/Q mixern beräknar I och Q signalerna utifrån följande uttryck:

$$I_{IF} = I_n \cos(2\pi\ f_{IF}) - Q_n \sin(2\pi\ f_{IF})$$
$$Q_{IF} = I_n \sin(2\pi\ f_{IF}) + Q_n \cos(2\pi\ f_{IF})$$

Ekvation 3: I/Q mixer

7.2.1.3 RF mixer

Sista länken i signalbehandlingskedjan är en analog blandare som används för att blanda upp IF signalen till radio frekvens.

7.2.2 Modifieringar av konstruktionen

Målet att jämföra FPGA realiseringen för den befintliga modulatorn med FPGA realiseringen för den modulatorn som konstruerats via Simulink och Xilinx System Generator gör att FPGA funktionen i de båda fallen bör vara samma i så stor utsträckning som möjligt. Detta krav är omöjligt att kombinera med att modulatorn skall implementeras på Xtreme DSP kortet eftersom inte samma hårdvara finns tillgänglig. Lösningen på detta blev att två olika versioner av modulatorn byggdes med hjälp av Simulink. En version där FPGA funktionen är identisk med den befintliga och en version där FPGA:n innehåller de funktioner som krävs för att implementera modulatorn på Xtreme DSP kortet.

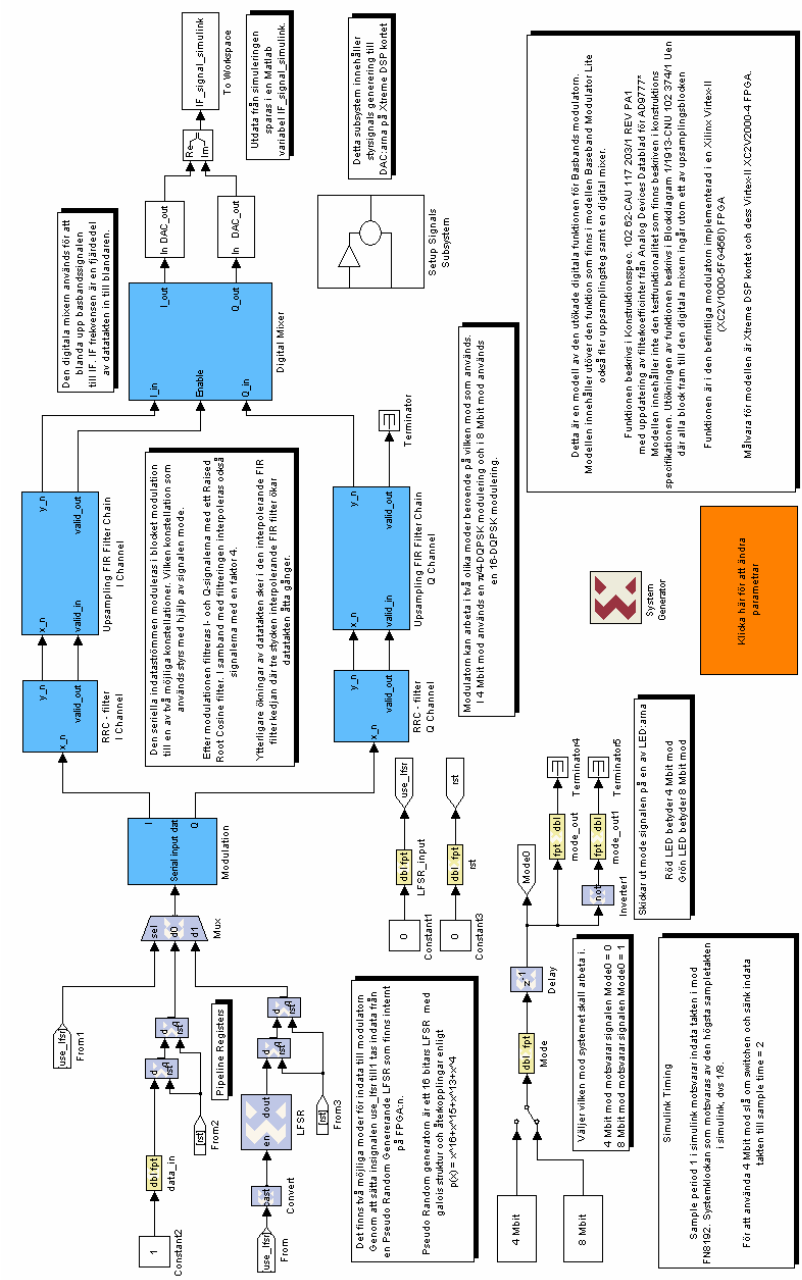
7.2.2.1 Baseband Modulator

För att implementera hela modulatorfunktionen på Xtreme DSP kortet där D/A omvandlarna inte innehåller interpolation och I/Q mixer modifierades FPGA funktionen till att även innefatta interpolationen och en mixer. Detta innebär att tre interpoleringssteg och en digital mixer lades till i FPGA funktionen jämfört med den befintliga FPGA:n. Det innebär också att datatakten i slutet av signalbehandlingskedjan är 147,456 MHz.

Vid en första syntesomgång visade syntesverktygen att denna klockfrekvens var för hög för den valda FPGA:n. På grund av detta togs ett av interpoleringsstegen bort och klocktakten skalades därmed ner till 73,728 MHz. Då strukturen på den digitala mixern som blandar upp basbandssignalen till IF är sådan att IF frekvensen är en fjärdedel av datatakten innebär detta också en nedskalning av IF frekvensen till 18,432 MHz.

För att mäta utsignalens spektrum har en pseudo random generator lagts till som möjlig indatakälla till modulatorn. Pseudo random generatorn realiseras med hjälp av ett återkopplat skift register så kallat LFSR, [18]. Pseudo random generatorn är intern på FPGA:n och aktiveras med insignalen lfsr_input.

Denna funktion modelleras i modellen Baseband Modulator



Figur 14: Blockschema Baseband Modulator

7.2.2.2 Baseband Modulator Lite

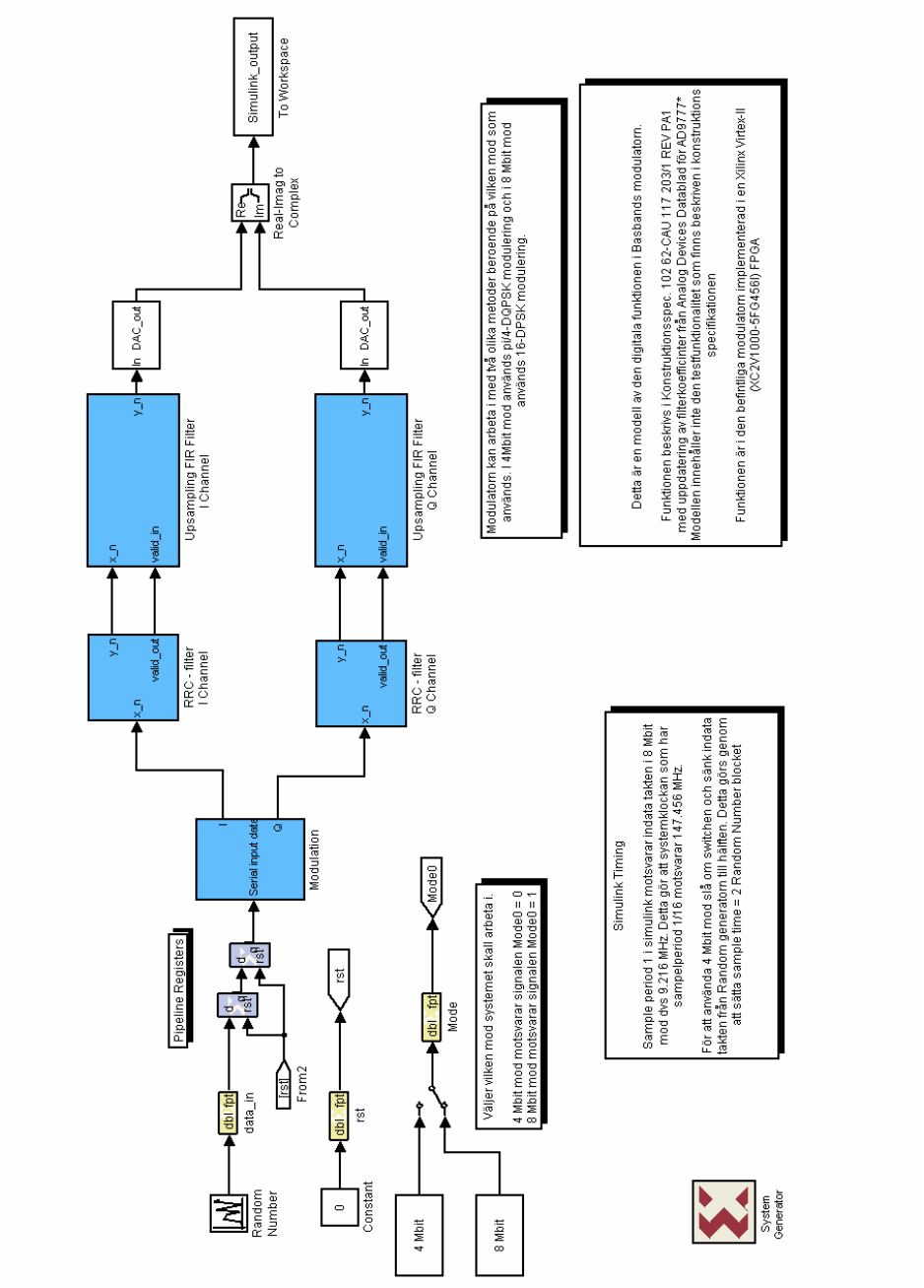
För att jämföra VHDL koden som skrivits för hand med den kod som Xilinx System Generator genererar bör FPGA funktionen vara identisk i de båda fallen. Därför byggdes modellen Baseband Modulator Lite. Lite indikerar att modellen är en mindre version av modellen Baseband Modulator (eng. Light).

Modellen utför samma uppgift som FPGA:n i den befintliga modultorn förutom en testfunktion som har utelämnats. FPGA:ns funktion beskrivs i avsnitt 7.2.1.1 Detta innebär att det inte varit aktuellt att realisera modellen för användning på Xtreme DSP kortet. Målet har istället enbart varit att jämföra resultatet av imlementationen baserat på de rapporter som genereras av implementationsverktygen. Av den anledningen har Xilinx Virtex-II XC2V1000-5 FPGA använts som målvara vid syntesen eftersom det är den FPGA som motsvarande handskriven kod är syntetiserad för.

Den testfunktion som utelämnats är så enkel att det inte bör påverka jämförelsen i någon större utsträckning att den har utelämnats i den genererade versionen.

Modellen Baseband Modulator Lite består av en digital modulator samt en efterföljande signalbehandling där signalen filtreras med ett root raised cosine filter och signalens datatakt ökas med ett interpolerande FIR filter. Ett blockschema för Baseband Modulator Lite visas i figur 15.

Modellen är uppbyggd för att en realisation som är identisk med den befintliga skall uppnås i så hög grad som möjligt.



Figur 15: Blockschema Baseband Modulator Lite

7.3 Implementation

Xilinx System Generator används för att generera VHDL kod för modellerna av modulatorn. Vid genereringen skapas också krav som används av syntesverktygen. Xilinx System Generator skapar också en testbänk och makron som kan användas för att simulera den generade VHDL koden.

För VHDL syntes används Synplify 7.1 från Synplicity, för FPGA implementation används verktyg som ingår i Xilinx ISE. ModelSim från Modeltech, Inc används för att simulera den generade VHDL koden och den slutgiltiga FPGA nätlistan.

På grund av de problem med synkronisering av flera klockdomäner som beskrivs i avsnitt 5.2.3.4 har den genererade VHDL koden modifierats för att korrekt funktion skall uppnås.

7.3.1 Baseband Modulator Lite

Koden som genereras av Xilinx System Generator har simulerats för att verifiera att funktionen är densamma som för Simulinkmodellen. Den resulterande nätlistan har också simulerats efter Place and Route.

7.3.1.1 Implementationsresultat

De olika verktygen genererar en mängd rapporter som beskriver resultatet av syntesen och implementationen. Bland annat rapporterar verktyget vilken maximal klockfrekvens som kan användas för FPGA:n.

Nedan följer utdrag ur PAR rapporten som beskriver hur stor del av FGPA:n som utnyttjas av baseband modulator lite.

```
Device utilization summary:

 Number of External IOBs           40 out of 324    12%
   Number of LOCed External IOBs    0 out of 40      0%

   Number of RAMB16s                3 out of 40      7%
   Number of SLICEs              1566 out of 5120   30%

   Number of BUFGMUXs               1 out of 16      6%


Overall effort level (-ol):   5 (set by user)
Placer effort level (-pl):    5 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    5 (set by user)
. . .

All constraints were met.

All signals are completely routed.
```

Figur 16: Utdrag ur PAR rapport för baseband modulator lite

7.3.1.2 Jämförelse med handskriven kod

För att få en uppfattning om hur bra den slutliga FPGA implementationen är har jag valt att titta på hur stor del av FPGA:n som utnyttjas. Jag har då valt att använda rapporterna från syntesverktyget för att jämföra resultatet.

```
Device utilization summary:

 Number of External IOBs          115 out of 324    35%
   Number of LOCed External IOBs  115 out of 115   100%

   Number of RAMB16s                1 out of 40      2%
   Number of SLICEs              1631 out of 5120   31%

   Number of BUFGMUXs               5 out of 16     31%


Overall effort level (-ol):   3 (set by user)
Placer effort level (-pl):    3 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    3 (set by user)
```

Figur 17: Utdrag ur PAR rapport för handskriven kod

Den siffra som anger hur stor del av FPGA:n som används är antalet slices. Fyra slices tillsammans utgör en CLB (Configurable Logic Block) som är den grundläggande byggstenen i en FPGA. Om man jämför de båda syntes resultaten ovan är utnyttjandet av FPGA:n nästan identiskt.

7.3.2 Baseband Modulator

Koden som genereras av Xilinx System Generator har simulerats för att verifiera att funktionen är densamma som för Simulinkmodellen. Den resulterande nätlistan har också simulerats efter Place and Route.

Syntesen har gjorts för den Xilinx Virtex-II XC2V2000-4 FPGA som sitter på Xtreme DSP kortet.

7.3.2.1 VHDL generering

Xilinx System Generator genererar förutom VHDL kod också information till syntesverktygen om vilka krav som olika delar av konstruktionen skall uppfylla. Dessutom kan man generera information till syntesverktygen om pin placering av de olika in- och utsignalerna i modellen. Detta gör att man inte behöver bry sig om VHDL koden förutom på en punkt. Man måste se till att den klocksignal som genereras av Xilinx System Generator mappas till en klockingång på FPGA:n.

7.3.2.2 Implementationsresultat

Vid implementationen genererar verktygen rapporten som beskriver hur väl de krav användaren ställer uppfylls av implementationen. Den maximala klockfrekvens som rapporteras av verktygen baseras på den längsta fördröjning som verktyget hittar i kretsen. Detta resultat kan vara missvisande om flera klockdomäner används i samma FPGA. System Generator genererar krav för att kontrollera timingen i varje klockdomän för sig och därmed räcker det att alla krav är uppfyllda för att kretsen skall kunna klockas med den hastighet som specificerats i Simulinkmodellen.

Nedan följer utdrag ur PAR rapporten som beskriver hur stor del av FGPA:n som utnyttjas av baseband modulator.

```
Device utilization summary:

 Number of External IOBs           46 out of 456    10%
   Number of LOCed External IOBs   46 out of 46    100%

   Number of RAMB16s                3 out of 56      5%
   Number of SLICEs              6184 out of 10752  57%

   Number of BUFGMUXs               1 out of 16      6%


Overall effort level (-ol):   5 (set by user)
Placer effort level (-pl):    5 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    5 (set by user)

. . .
All constraints were met.

All signals are completely routed.
```

Figur 18: Utdrag ut PAR rapport för baseband modulator

Utnyttjandet av FPGA:n ges av antalet utnyttjade slices. Att 6184 slices eller 57 % av FPGA:n utnyttjas är en relativt hög utnyttjandegrad. För Baseband Modulator Lite är motsvarande siffra 1566 slices. Denna ökning av FPGA utnyttjandet förklaras med strukturen hos de filter som avslutar signalbehandlingskedjan. För att hålla samma interna klockfrekvens då indatatakten till filtret ökas måste filtret implementeras med en mer parallell struktur.

Till exempel har det sista filtret i kedjan en indatatakt på halva systemklockfrekvensen. För att filterkärnan ska kunna arbeta på systemklockfrekvensen krävs att åtta parallella beräkningar kan göras för att de 16 bitarna i insignalen ska kunna behandlas under en klockcykel.

7.4 Verifiering

För att verifiera funktionen hos modulatorn har mätningar gjorts på Xtreme DSP kortet. MATLAB har använts för att analysera utsignalen från kortet för att se hur den modulerade signalen ser ut.

7.4.1 Mätutförande

Ett problem som uppstod vid mätningarna var att hitta en stabil klockgenerator. Vid ett första försök användes en pulsgenerator för att generera en klocka med frekvensen 73,728 MHz. Det visade sig dock att stabiliteten hos denna klocka inte var tillräckligt hög. För att lösa detta användes istället den kristall oscillator som finns monterad på Xtreme DSP kortet. Det medförde en sänkning av klockfrekvensen till 65 MHz. Detta innebar en skalning av IF frekvensen till 16,25 MHz.

Som indata har en pseudo random sekvens som genererats av den inbyggda pseudo random generatorn använts. Då den inte har något stöd för att växla datatakt har samma indatatakt använts i båda moderna. Indatatakten har varit 8,125 MHz. Detta innebär att endast hälften av indata samplen moduleras i 4 Mbit/s moden.

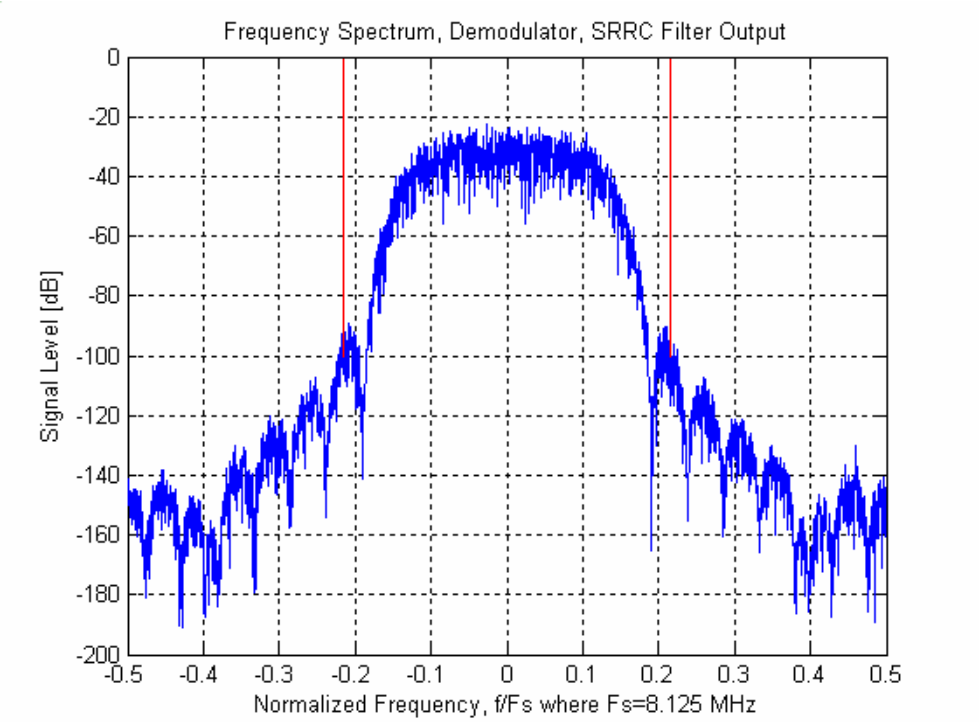
För att analysera resultatet mättes utsignalen från D/A omvandlarna med ett samplings oscilloskop. Utsignalerna samplades med 250 Msampel/s under 400 µs, vilket gav en datamängd om 100 000 sampel.

Data insamling gjordes i båda moderna för att verifiera funktionen.

7.4.2 Analys av mätdata

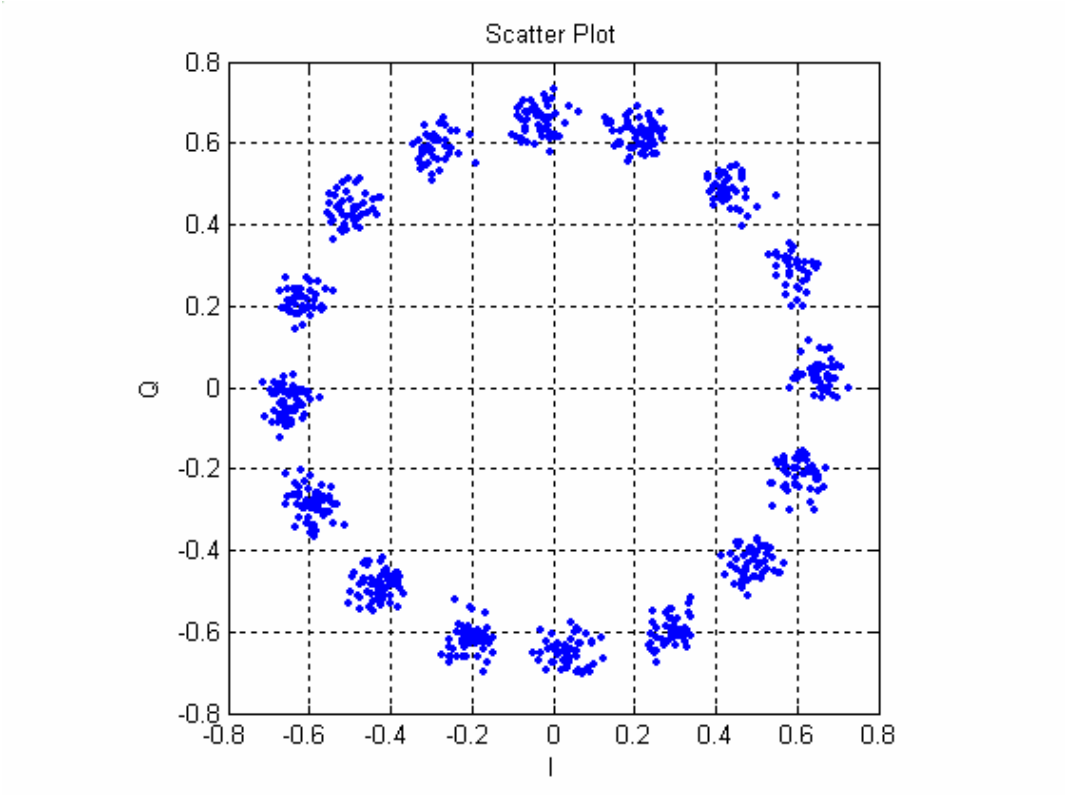
För att analysera de uppmätta utsignalerna från modulatorn användes MATLAB. Det script som använts för att analysera mätdata har kopierats från utvärderingen av den befintliga konstruktionen och har använts med små modifikationer för att passa den något annorlunda datatakten. Först blandades signalen ner till basband och en basbands filtrering genomfördes för att filtrera bort eventuella oönskade frekvenskomponenter som introducerats vid sampling och blandning. Signalen decimeras sedan till 8,125 MHz vilket ger fyra sampel per symbol. Den decimerade signalen filtreras genom ett root raised cosine filter

Spektrum för utsignalen från root raised cosine filtret har plottats i figur 19. I figuren har också en bandbredd om 3,5 MHz indikerats.

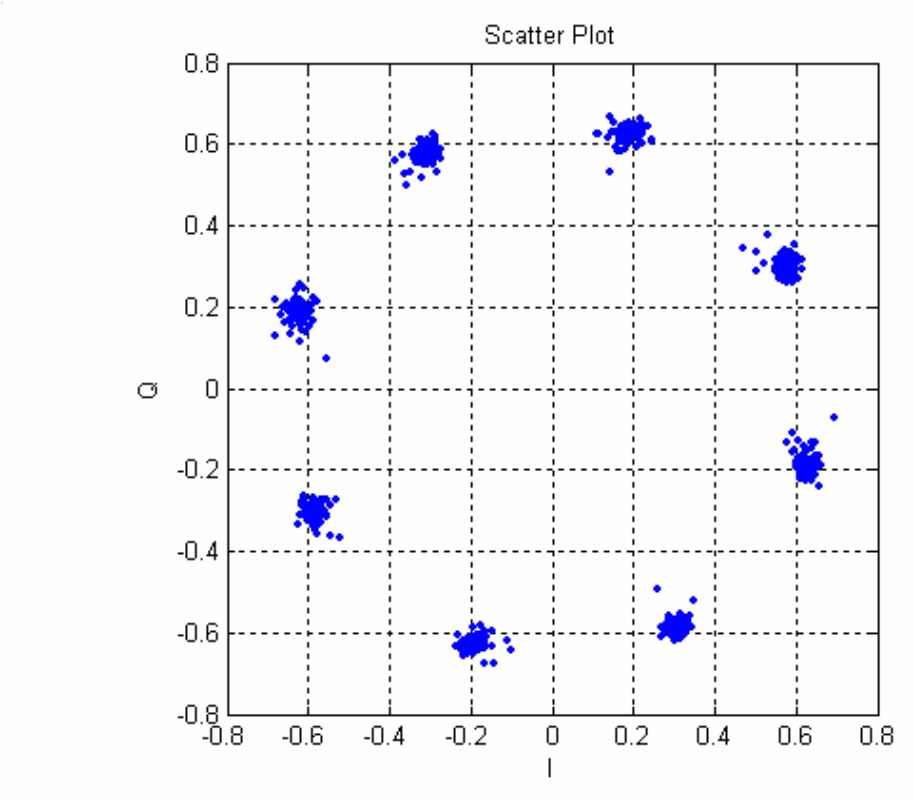


Figur 19: Spektrum för signalen i 8 Mbit/s mod

För att verifiera funktionen hos modulatorn har symbolkonstellationerna i de olika moderna undersökts. Konstellationen fås genom att man plottar I- mot Q-signalen för en viss symbol. Eftersom det efter decimeringen finns fyra sampel per symbol är det bara var fjärde sampel som har plottats.



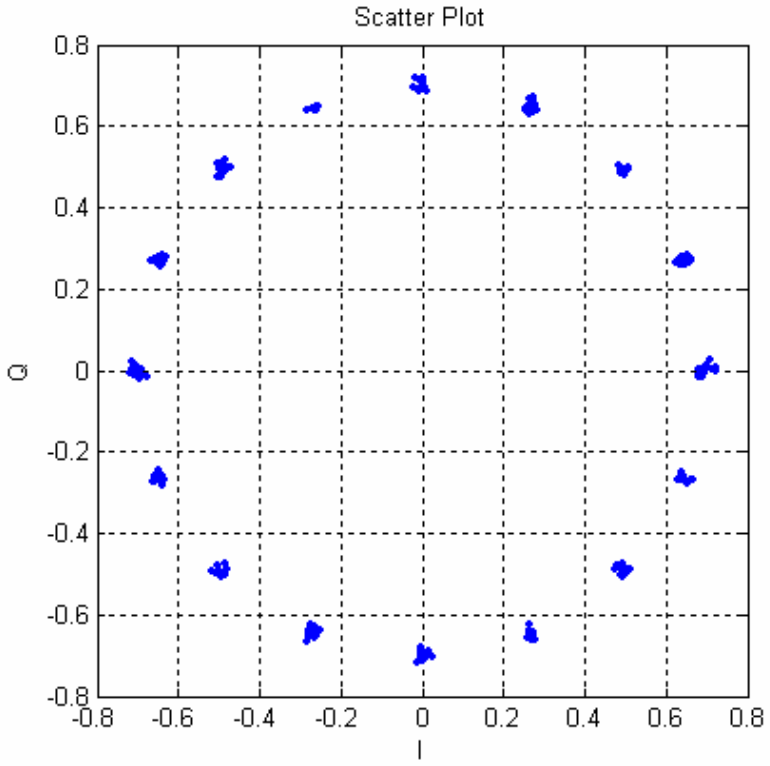
Figur 20: Uppmätt symbolkonstellation för 8 Mbit/s mod



Figur 21: Uppmätt symbolkonstellation för 4 Mbit/s mod

De uppmätta konstellationerna innehåller en del brus och även en ganska tydlig fasvridning men man kan tydligt se de förväntade konstellationerna. Fasvridningen uppstår på grund av en liten skillnad i den verkliga klockfrekvensen i FPGA:n och den som antas i MATLAB scriptet.

Motsvarande figurer har även plottas vid simulering av modellen.



Figur 22: Simulerad symbolkonstellation för 8 Mbit/s mod

Även den simulerade konstellationen innehåller en del brus. Detta kan förklaras med kvantiseringen av signalerna i modellen. Observera att antalet symboler som plottats skiljer sig mellan de tre figurerna ovan.

Med utgångspunkt från figurerna ovan dras slutsatsen att modellen har kunnat föras över till hårdvara utan att några fel i konstruktionen har uppstått.

7.5 Resultat

Jämförelsen av FPGA utnyttjandet mellan den kod som genererats med Xilinx System Generator respektive skrivits för hand talar starkt för att Xilinx System Generator mycket väl kan konkurrera med handskriven VHDL kod. Detta är ett mycket viktigt resultat eftersom det visar att man inte behöver använda större och därmed dyrare FPGA:er om man utnyttjar Xilinx System Generator. Man kan också använda Xilinx System Generator för att generera kod till uppgraderingar av redan befintliga konstruktioner.

En reservation vid analysen av resultatet kan vara på sin plats då funktionen hos den FPGA implementation som jämförts inte har verifierats direkt. Däremot ingår Baseband Modulator Lite i den utökade FPGA implementationen som har verifierats i hårdvara. Detta kan anses vara verifiering nog av funktionen hos den kod som har jämförts men för att vara helt säker skulle man kunna ladda FPGA:n på det befintliga kortet med den genererade koden och genomföra mätningar. Jag har dock valt att inte lägga ner det arbete som skulle krävas för att utföra denna mätning. Jag anser att den verifiering som gjorts klart visar att resultatet av jämförelsen är relevant.

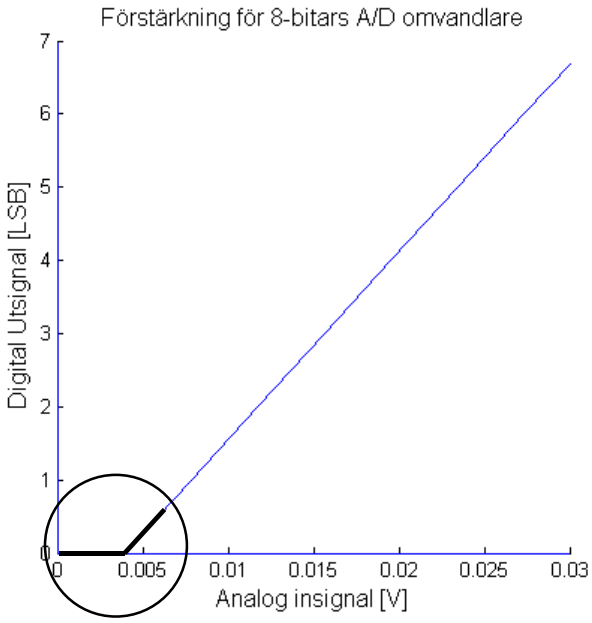
Det har också visat sig vara möjligt att använda ett massproducerat utvecklingskort för att bygga en prototyp av en digital signalbehandlings funktion. Utvecklingstiden från en färdig konstruktionsspecifikation kan drastiskt förkortas om en färdig hårdvaruplattform kan användas. Om det dessutom är möjligt att "off-the-shelf" köpa en plattform, som Xtreme DSP kortet, kan man spara både tid och pengar på att man inte behöver utveckla en egen plattform.

8 Automatic Gain Control

För att ytterligare testa Xilinx System Generator i ett annat testfall än modulatorn används ett exempel med en automatic gain control (AGC) funktion. Funktionen har realiserats med hjälp av Xtreme DSP kortet och en extern dämpare har använts.

8.1 Bakgrund

Anledningen till att denna funktion valts som testfall är att funktionen bygger på kontroll av en signalbehandling baserad på styrsignaler och därmed ställer större krav på kontroll och styrning än modulatorn.



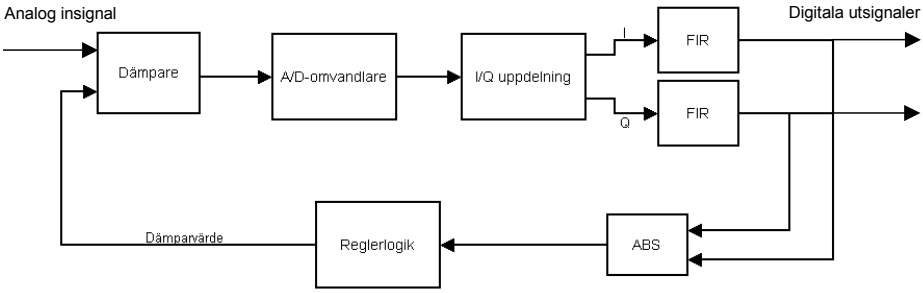
Figur 23: Olinjäritet hos A/D-omvandlarens förstärkning

AGC är ett exempel på en funktion som ingår i de radarmottagare som Ericsson Microwave producerar. Målet med AGC funktionen är att reglera brusnivån på ingången till A/D-omvandlaren i mottagaren.

Anledningen till att man vill reglera brusnivån på ingången till A/D-omvandlaren är att man vill använda bruset för att maskera A/D-omvandlarens olinjära förstärkning vid 0 V insignal, se figur 23. Genom att se till att bruset lyfter upp insignalen till en nivå något över 0 V kan man garantera att insignalen använder det linjära området av A/D-omvandlarens förstärkning.

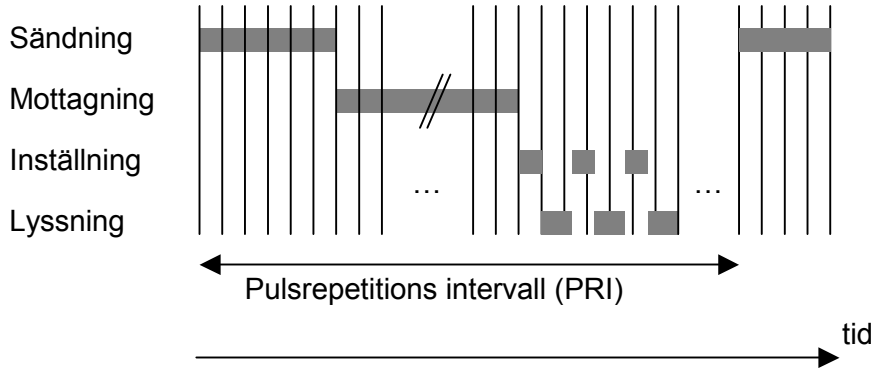
8.2 Beskrivning AGC funktion

Målet för AGC loopen är att reglera brusnivån på ingången av A/D-omvandlaren. En översikt av AGC funktionen visas i figur 24. Regleringen görs med hjälp av en dämpare. Genom att efter I/Q uppdelning och FIR filtrering av den mottagna radarsignalen uppskatta väntevärdet hos signalen kan brusnivån på ingången av A/D-omvandlaren uppskattas. För att utföra mätningen utnyttjar man ett tidsintervall då ingen signal tas emot av antennen. Uppskattningen görs genom att man beräknar en approximation av I/Q-vektorns absolutbelopp. Absolutbeloppet jämförs sedan med ett beräknat tröskelvärde och dämparen styrs utifrån antalet tröskelöverskridanden.



Figur 24: Översikt AGC funktion

Tidsdiagrammet i figur 25 indikerar de viktigaste tidslägena i ett radarsystem som har betydelse för AGC funktionen. Under sändning skickas en puls ut från antennen. Pulsens eko tas emot under mottagningsfasen. Efter mottagning sker lyssning för att bestämma brusnivån. Mellan lyssningarna kan inställning göras för att t ex ställa in olika mottagningsfrekvenser vid vilka man vill mäta brusnivån. Proceduren med utsändning, mottagning och brusmätning upprepas med en period som kallas pulsrepetitionsintervall, förkortas PRI.



Figur 25: Tidsdiagram för radarfunktion

Varje PRI är i sin tur uppdelat i ett antal fållor. Längden på en fålla motsvarar avståndsupplösningen för radarsystemet.

8.2.1 Beräkningar och bakgrund

Vid styrningen av dämparen används absolutbeloppet av I/Q-vektorn för att uppskatta brusnivån på ingången av A/D-omvandlaren. Mätningen görs genom att en summa av sexton absolutbelopp bildas. Denna summa jämförs med ett tröskelvärde för att avgöra om dämparen skall justeras eller inte. För att minska känsligheten för störning av mottagaren uppdateras dämparen baserat på ett antal mätningar och en tabell används för att överföra antalet tröskelöverskridanden till en dämparkorrektion. Om många mätningar används för att styra dämparen hinner antennen rotera flera varv och en störare som endast stör från en riktning har inte så stor inverkan på regleringen.

8.2.1.1 Filtrens inverkan på brusnivån

Eftersom brusmätningen görs efter I/Q uppdelningen och FIR filtreringen behöver man veta hur I/Q uppdelningen och FIR filtreringen påverkar brusnivån. Nedan följer ett exempel på hur dessa filter kan påverka brusnivån.

I/Q uppdelningen görs med hjälp av två FIR filter vars överföringsfunktioner ges av ekvation 5.

$$I(z) = 1 - 11z^{-2} + 15z^{-4} - 5z^{-6}$$
$$Q(z) = 5z^{-1} - 15z^{-3} + 11z^{-5} - z^{-7}$$

Ekvation 4: Överföringsfunktion för I/Q filter

Den I/Q uppdelade signalen filtreras med ett pulsanpassat FIR filter. Det pulsanpassade filtret används för att integrera sex sampel. Samma filter används på både I- och Q-kanalen.

$$H(z) = 1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} - z^{-10}$$

Ekvation 5: Överföringsfunktion för pulsanpassat FIR filter

Amplitudförstärkningen för signal i ett FIR filter ges av ekvation 6.

$$A_S = \sum_{n=0}^{N} |h(n)|$$

Ekvation 6: Amplitudförstärkning för signal

Amplitudförstärkningen för signalen i I/Q filtren och det pulsanpassade FIR filtret beräknas.

$$\begin{cases} A_S = 1+11+15+5 = 32 & I(z) \\ A_S = 5+15+11+1 = 32 & Q(z) \\ A_S = 1+1+1+1+1+1 = 6 & H(z) \end{cases}$$

Ekvation 7: Signalförstärkning för filtren i AGC funktionen

Amplitudförstärkningen för vitt brus i ett FIR filter ges av ekvation 8.

$$A_N = \sqrt{\sum_{n=0}^{N} h(n)^2}$$

Ekvation 8: Amplitudförstärkning för vitt brus

Ekvation 8 anger brusförstärkningen för vitt brus. Bruset på ingången till det pulsanpassade FIR filtret har filtrerats genom I/Q filtren och kan inte betraktas som vitt brus. För att beräkna brusförstärkningen för det pulsanpassade filtret används det totala filtret som fås genom att kaskadkoppla I/Q filtren med det pulsanpassade filtret.

Överföringsfunktionen för kaskadkopplade filter fås genom att multiplicera de z-transformerade överföringsfunktionerna för de individuella filtren som ingår i kaskadkopplingen, [19].

$$I(z)*H(z)=1-12z^{-2}+27z^{-4}-32z^{-6}+32z^{-8}-32z^{-10}+31z^{-12}-20z^{-14}+5z^{-16}$$
$$Q(z)*H(z)=5z^{-1}-20z^{-3}+31z^{-5}-32z^{-7}+32z^{-9}-32z^{-11}+27z^{-13}-12z^{-15}+z^{-17}$$

Ekvation 9: Överföringsfunktion för totalt filter

Ekvation 8 kan användas för att beräkna brusförstärkningen för I/Q filtren och det totala filtret.

$$A_N=\sqrt{1^2+11^2+15^2+5^2}=19{,}287...$$

Ekvation 10: Brusförstärkning för I/Q filter

$$A_N=\sqrt{1^2+12^2+27^2+32^2+32^2+32^2+31^2+20^2+5^2}=73{,}021...$$

Ekvation 11: Brusförstärkning för total filtrering

Genom att beräkna kvoten mellan den totala brusförstärkningen och brusförstärkningen för I/Q filtren fås brusförstäkningen för det pulsanpasade FIR filtret.

$$A_N=\frac{73{,}021}{19{,}287}=3{,}786$$

Ekvation 12: Brusförstärkning för pulsanpassat FIR filter

För att minska komplexiteten på filterrealisationen skalas data mellan I/Q uppdelningen och det pulsanpassade filtret för att minska antalet bitar i datarepresentationen. Utdata från A/D-omvandlaren representeras med 12 bitar. Det ger ett indataområde på ±2048 LSB. Utdata från I/Q uppdelningen har förstärkts med A_S = 32 vilket ger utdata i området ±65536 LSB.

Brusnivån på ingången till I/Q uppdelningen skall vara 1,6 LSB. Bruset förstärks med A_N = 19,287 vilket ger en brusnivå på utgången av I/Q uppdelningsfiltret som motsvarar 30,859 LSB

Om ingen skalning av utsignalen görs kommer indata till det pulsanpassade filtret representeras av 17 bitar där ungefär de fem minst signifikanta bitarna består av vitt brus. Genom att skala utsignalen med 2^{-4} minskas utdataområdet till ±4096 LSB samt brusnivån minskas till 1,929 LSB. Indata till det pulsanpassade filtret kan nu representeras med 13 bitar. Detta förenklar realisationen av det pulsanpassade FIR filtret. Skalningen kräver ingen extra hårdvara då den endast består i att binärpunkten flyttas och de fyra minst signifikanta bitarna kastas.

För det pulsanpassade filtret är A_S = 6 vilket ger utdata i området ±24576 LSB. Bruset från ingången förstärks med A_N = 3,786 vilket ger 7,303 LSB brus på utgången. Genom att skala utgången med 2^{-2} fås utdata i området ±6144 LSB och en brusnivå ut från filtret som motsvarar 1,826 LSB. Utdata från filtret representeras med 14 bitar.

Resultatet är att en brusnivå på 1,6 LSB på ingången till A/D-omvandlaren motsvaras av en brusnivå på 1,826 LSB på utgången av det pulsanpassade FIR filtret.

8.2.1.2 Sannolikhetsfördelning för brus

Bruset på ingången till A/D omvandlaren modelleras med vitt gaussiskt brus med väntevärde μ = 0 och varians σ^2.

Frekvensfunktionen för vitt gaussiskt brus:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$ [21] sid 418

Ekvation 13: Frekvensfunktion för vitt gaussiskt brus

Vid beräkningen av I/Q-vektorns absolutbelopp förändras brusets sannolikhetsfördelning från normalfördelning till Rayleigh fördelning, [20].

$$f_r(x) = \begin{cases} \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Ekvation 14: Frekvensfunktion för Rayleigh fördelat brus

Förändringen av brusets fördelning leder till att väntevärdet för bruset förändras. Väntevärdet för det Rayleigh fördelade bruset beräknas enligt:

$$\mu_r = \int_{x=0}^{\infty} x\, f_r(x)\, dx = \sigma\sqrt{\frac{\pi}{2}}$$ [21] sid 419

Ekvation 15: Väntevärde för Rayleigh fördelat brus

8.2.1.3 Beräkning av tröskelvärde för brusmätning

Den önskade brusnivån 1,6 LSB på ingången av A/D-omvandlaren motsvaras av att σ för bruset på ingången av A/D-omvandlaren är 1,6 LSB. Efter beloppsbildningen motsvarar denna standardavvikelse ett väntevärde för det Rayleigh fördelade bruset.

Vid I/Q-uppdelning och FIR filtrering sker en skalning av σ för bruset enligt avsnitt 8.2.1.1.

Tröskelvärdet vid mätningen av brusnivån motsvarar väntevärdet för absolutbeloppet då ingen signal tas emot av mottagaren.

$$\mu = \sigma\ efter\ \ filtrering * \sqrt{\frac{\pi}{2}} = 1{,}826 LSB * \sqrt{\frac{\pi}{2}} = 2{,}289...LSB$$

Ekvation 16: Väntevärde för absolutbeloppet då ingen signal tas emot

Sexton beloppssampel summeras vid mätningen av brusnivån. Detta ger följande tröskelvärde.

$$\text{Tröskelvärde } = 16\mu = 16 * 2{,}289\ LSB = 36{,}612\ LSB$$

Ekvation 17: Tröskelvärde för mätning av brusnivån

Resultatet är att brusnivån 1,6 LSB på ingången av A/D-omvandlaren motsvaras av tröskelvärdet 36,612 LSB vid brusmätningen.

8.2.2 Modifiering av AGC funktionen

En modell av AGC funktionen har konstruerats i Simulink. Modellen har använts för att generera VHDL kod som använts för att realisera modellen i FPGA:n på Xtreme DSP kortet.

För att underlätta verifieringen av funktionen i hårdvara har vissa modifieringar gjorts till AGC funktionen. Den största förändringen är att en sinussignal har använts istället för brus som analog insignal vid mätningarna. Detta medför att beräkningarna för filtrens inverkan på signalnivån och beräkningen av tröskelvärdet har gjorts om för att passa mätningarna av en sinussignal.

För att mätningarna skall underlättas har börvärdet för signalnivån in till A/D omvandlaren satts till 50 mV toppvärde för sinussignalen. A/D omvandlarna på Xtreme DSP kortet använder 14 bitar och har ett insignalsområde mellan -1 V och 1 V. Av dessa 14 bitar används de 12 minst signifikanta i AGC funktionen. 50 mV motsvarar då 410 LSB efter A/D omvandlingen.

8.2.2.1 Filtrens inverkan på signalnivån

De beräkningar som redovisas i avsnitt 8.2.1.1 har modifierats för att gälla då nivån av en sinussignal skall regleras istället för en brusnivå. De modifierade beräkningarna redovisas nedan.

Vid I/Q-uppdelning av en sinussignal fås konstant absolutbelopp för I/Q vektorn. Absolutbeloppet av I/Q vektorn motsvarar toppvärdet för den I/Q-uppdelade sinus signalen.

Vid mätningarna har skalning med 2^{-1} använts efter det pulsanpassade FIR filtret.

Utsignalen från I/Q-uppdelning och FIR filtrering vid 410 LSB insignal från A/D omvandlaren beräknas enligt:

A_S I/Q-uppdelningsfilter = 32 Skalning efter I/Q filter = 2^{-4}

A_S FIR filter = 6 Skalning efter FIR filter = 2^{-1}

Utsignal = 410 LSB * 32 * 2^{-4} * 6 * 2^{-1} = 2460 LSB

Tröskelvärdet för 16 absolutbeloppssampel är
$16 * 2460\ LSB = 39360\ LSB$.

8.3 Implementation

Xilinx System Generator användes för att generera VHDL kod från modellen av AGC funktionen. Synplify 7.1 användes för VHDL syntes och verktyg från Xilinx ISE användes för FPGA implementation. Den genererade VHDL koden och den resulterande nätlistan simulerades båda med hjälp av ModelSim.

På grund av de problem med synkronisering av flera klockdomäner som beskrivs i avsnitt 5.2.3.4 har den genererade VHDL koden modifierats för att undvika problem vid reset av FPGA:n.

8.3.1 Implementationsresultat

Syntesen av VHDL koden genererade följande utnyttjande av FPGA:n enligt place and route rapporten.

```
Device utilization summary:

 Number of External IOBs           70 out of 456    15%
   Number of LOCed External IOBs   70 out of 70    100%

   Number of RAMB16s                1 out of 56      1%
   Number of SLICEs              1235 out of 10752  11%

   Number of BUFGMUXs               1 out of 16      6%



Overall effort level (-ol):   3 (set by user)
Placer effort level (-pl):    3 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    3 (set by user)
. . .

All constraints were met.

All signals are completely routed.
```

Figur 26: Utdrag ur PAR rapport för ACG funktionen

Endast 11% av FPGA:ns slices används av AGC funktionen. Eftersom AGC funktionen endast är en del av en större mottagarfunktion och resurser krävs för de övriga delarna av mottagaren är detta resultat nödvändigt.

8.4 Verifiering

För att verifiera funktionen hos AGC funktionen har mätningar gjorts på Xtreme DSP kortet. För att underlätta mätningarna har vissa delar av funktionen ändrats. Dessa modifikationer beskrivs i avsnitt 8.2.2.

8.4.1 Mätutförande

Vid verifiering av AGC funktionen har mätningar av de resulterande I- och Q-signalerna gjorts. Genom att AGC funktionen reglerar insignalen till I/Q-uppdelningen motsvaras en fungerande AGC funktion av att nivån på I- och Q-signalerna är konstant. Dock sker en skalning av signalnivån genom filtren enligt 8.2.2.1. Utsignalen från FIR filtreringen vid 50 mV (410 LSB) insignal till A/D omvandlaren motsvarar 2460 LSB.

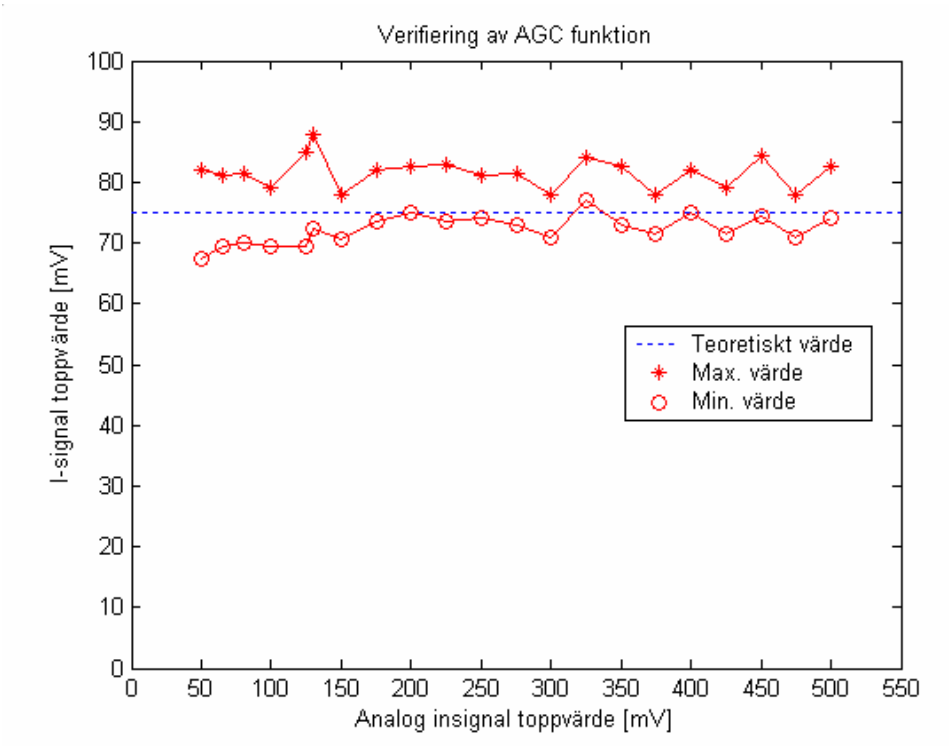
På grund av att antalet digitala utgångar från Xtreme DSP kortet är begränsat används de två analoga utsignalerna för att mäta I- och Q-signalerna. D/A-omvandlarna använder 14 bitar och omvandlingen från 16 bitars representation av I- och Q-signalerna internt medför en skalning med 2^{-2}. Denna skalning medför att 410 LSB insignal motsvarar 2460 / 4 = 615 LSB utsignal. Denna digitala signal motsvaras efter D/A omvandlingen av 75 mV analog utsignal.

Den dämpare som använts vid mätningarna har en upplösning på 1 dB. På grund av den begränsade upplösningen hos dämparen kan inte en exakt reglering ske. Istället leder regleringen till att dämparvärdet slår mellan två värden som ger I- och Q-nivåer kring den teoretiska tröskeln.

8.4.2 Resultat

För att verifiera reglerfunktionen hos AGC:n har mätningar av I- och Q-signalernas spänning gjorts. En korrekt fungerande AGC funktion medför att toppvärdet av de sinusformade I-och Q-signalerna skall vara 75 mV.

I figur 27 visas toppvärdet för I-signalen för olika insignalsamplituder. Varje insignal ger två nivåer i diagrammet för de två olika dämparvärden som regleringen växlar mellan på grund av dämparens begränsade upplösning.



Figur 27: Verifiering av AGC funktion

Mätningarna indikerar att den verkliga reglerade signalnivån är något högre är den beräknade. En av orsakerna till detta är att approximationen av absolutbeloppet har ett fel som gör att reglerloopen mäter en något för låg signalnivå och därmed kommenderas ett lägre dämparvärde än idealt.

8.5 Resultat

Mätningarna visar att realiseringen av AGC funktionen på Xtreme DSP kortet med hjälp av Simulink och System Generator är möjlig. Genom att arbeta med Simulink och System Generator höjs abstraktionsnivån för konstruktören som inte längre har möjlighet att styra exakt hur VHDL koden ska se ut. De krav som AGC funktionen ställer på möjlighet att exakt styra när sampling skall ske och när mätningar skall göras går dock att uppfylla trots att konstruktionen sker på en högre abstraktionsnivå.

Det visar sig också vara möjligt att använda Xtreme DSP kortet för att realisera den huvudsakliga digitala funktionen i AGC:n. Det har varit möjligt att använda Xtreme DSP kortet tillsammans med en dämpare för att verifiera hela AGC funktionen.

9 Datorstödd elektronikkonstruktion

Allt eftersom elektroniska system blir allt mer komplexa ökar kraven på bra verktyg för elektronikkonstruktion. Dagens elektronikkonstruktörer brottas inte bara med krav på hög prestanda utan också med krav på låg effektförbrukning, låga kostnader och kort utvecklingstid. Därmed blir datorverktyg och metoder för datorstödd utveckling allt viktigare vid utveckling av elektroniska system, [1].

Den snabba utvecklingen av FPGA:er med finare processer (ner till 90 nm) och lägre effektförbrukning har dessutom gjort FPGA:er till möjliga plattformar för system on chip (SoC) tillämpningar.

9.1 IP baserad konstruktion

Både Xilinx och Altera satsar på IP baserade lösningar för att göra FPGA:er mer attraktiva för DSP konstruktörer som traditionellt inte har erfarenhet av digitalkonstruktion på FPGA nivå. Xilinx System Generator är en del i detta initiativ från Xilinx, [22].

De flesta IP block inom signalbehandling är telekominriktade. Detta syns också i Xilinx blockset där många av de mer avancerade blocken är inriktade mot kommunikationstillämpningar.

Utvecklingen av större FPGA:er med fler specialiserade funktioner gör att marknaden för IP lösningar ökar. Större FPGA:er ger möjligheter för SoC tillämpningar som tidigare bara var möjliga genom att en ASIC konstruerades, [23].

9.1.1 Högre abstraktionsnivå

Genom att IP kärnor används i högre utsträckning ökar abstraktionsnivån för elektronikkonstruktörer. Genom användning av färdiga IP block minskar behovet av konstruktion av funktioner på mycket låg nivå.

Genom att abstraktionsnivån ökar får delsystemkonstruktörer mer inflytande över den slutgiltiga konstruktionen. De beslut som fattas på system- eller delsystemnivå får genom den allt mer automatiserade konstruktionsprocessen större och större inflytande på den slutliga realiseringen. Detta ställer krav på delsystemkonstruktören som måste vara medveten om hur hans beslut påverkar den slutliga realiseringen.

9.2 Användning av MATLAB och Simulink

MATLAB och Simulink kan erbjuda möjligheter att simulera ett signalbehandlingssystem. Utveckling av funktioner för att implementera en modell gjord i Simulink i t ex en FPGA eller en digital signal processor har dessutom gjort Simulink till ett alternativ för konstruktion.

9.2.1 Systemutveckling

Genom att använda MATLAB och Simulink för att göra en modell av systemet som sedan kan användas för implementation med hjälp av Xilinx System Generator kan flera fördelar utnyttjas. Idealt skulle en arbetsmetod med flera modeller på olika abstraktionsnivå kunna användas vid konstruktionsarbetet.

Den ideala arbetsmetoden använder en Simulinkmodell av systemet som är central i hela utvecklingsarbetet. Modellen av systemet är i början av utvecklingsfasen en högnivåmodell av de algoritmer som skall implementeras. Denna modell innehåller inte detaljerad information om hur funktionen realiseras.

Genom att bryta ner modellen till mer och mer detaljerade delsystem kan en realisering av funktionen uppnås. Delsystemkonstruktören har under hela arbetet systemmodellen som referens i sitt arbete och kan kontrollera att delsystemet uppfyller de krav som ställs för att kraven på hela systemet skall uppfyllas.

Det slutliga steget i nedbrytningen är att modellen endast består av block som ingår i Xilinx blockset. För denna nedbrytning krävs kompetens inom digitalkonstruktion för att uppnå en högpresterande realisering.

För att nå den slutliga implementationen krävs dessutom kunskaper om hur VHDL kod syntetiseras och implementeras. För detta arbete krävs kompetens inom digitalkonstruktion, FPGA användning och VHDL implementation.

9.2.1.1 Fördelar

Genom att en modell är central i hela konstruktionsprocessen kan kommunikationen mellan de medarbetare som är inblandade i projektet underlättas. Modellen kan användas som referens och risken för missförstånd minskas. Genom modellen är alla medarbetare knutna till en gemensam utvecklingsmiljö vilket också underlättar arbetet.

Nedbrytningen av modellen till realiserbara delsystem underlättas om en referensmodell kan användas för att simulera delsystemet som en del av hela systemet. Genom att systemet kan simuleras under hela konstruktionsprocessen ökar chansen att implementeringen lyckas utan tidsödande felsökning.

Utvecklingstiden kan förkortas genom att Xilinx System Generator används för att generera VHDL kod som kan användas för att implementera modellen. Genom att System Generator används höjs abstraktionsnivån och mindre tid läggs på lågnivåkonstruktion.

9.2.1.2 Begränsningar

Den modell som byggs upp av systemet är en funktionell modell. Vid systemkonstruktion finns många krav på systemet som inte direkt har med funktionen att göra t ex systemets storlek, vikt och det temperaturområde inom vilket korrekt funktion måste kunna garanteras. Dessa aspekter av konstruktionen kan inte inkluderas i en Simulinkmodell.

Verktyg som Xilinx System Generator realiserar en modell i en FPGA. De delar av systemet som skapar förutsättningar för FPGA:n t ex spänningsmatning och minne för att ladda FPGA:n ingår inte i modellen. Dessa delar av systemet måste fortfarande konstrueras på traditionell väg.

9.2.2 Modellering

Det är enkelt och går fort att komma igång och bygga modeller med Simulink. Både MATLAB och Simulink innehåller mycket dokumentation om de funktioner som finns till användarens förfogande. Det krävs dock kunskap om det som skall modelleras. Trots att Simulink innehåller många block som utför mycket kraftfulla funktioner måste användaren kunna parametrisera blocken för att korrekt funktion skall modelleras.

9.2.2.1 Användarvänlig miljö

Simulink erbjuder en användarvänlig modelleringsmiljö. Modellens uppbyggnad som ett blockschema gör att strukturen är tydlig. Genom att använda subsystem blir modellerna enkla att överblicka och förstå.

9.2.2.2 Enkel kontroll av funktionen

Det är enkelt att lägga till block för att kontrollera data i en viss punkt i modellen. Detta är en stor fördel jämfört med VHDL simulatorer där VHDL koden måste kompileras innan den kan simuleras. Genom att System Generator kan användas för att skapa VHDL kod som har samma funktion som modellen behöver endast simulering av den genererade koden göras för att verfiera att beteendet är detsamma som modellens.

9.2.2.3 Analys av data

MATLAB innehåller funktioner för att snabbt och enkelt analysera data från simuleringarna i Simulink. Utdata kan sparas i MATLABs arbetsyta och funktioner för t ex spektralanlys kan användas.

Genom att använda MATLAB kan data från simuleringen dessutom jämföras med mätdata från den slutliga implementationen.

9.2.3 Simuleringstid

En viktig egenskap för simuleringsverktyg är hur lång tid simuleringen tar. För att simuleringar av större system skall vara praktisk genomförbara kan inte simuleringstiden vara för lång.

En modell som används för att generera VHDL måste vara detaljerad för att genereringen skall vara möjlig. Om en mycket detaljerad modell av ett stort system skall simuleras kan simuleringstiden bli mycket lång.

9.2.3.1 Simuleringstid AGC

Simuleringstiden visade sig vara ett problem vid simulering av AGC funktionen. För att AGC funktionen skall vara mindre känslig för störning görs många mätningar innan dämparen uppdateras. I de radarmottagare som använder AGC funktionen uppdateras dämparen ungefär var femte sekund. Fem sekunder motsvarar $\approx 200 \cdot 10^6$ klockcykler för modellen.

Simulering av 100 000 klockcykler för modellen tar ca 3 minuter att genomföra. Detta innebär att det skulle dröja ca 4 dygn mellan dämparuppdateringarna i en simulering av den fullständiga AGC funktionen. Detta gör simulering av AGC funktionen i sin helhet omöjlig i praktiken.

9.2.3.2 Hårdvaruaccelererad simulering

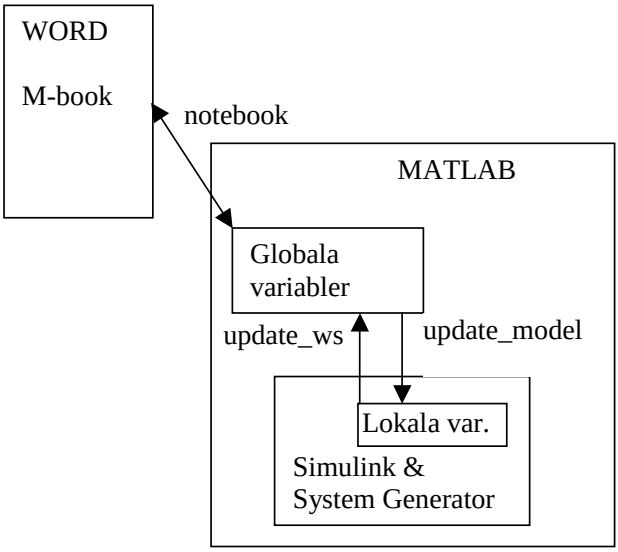
För att förkorta simuleringstiden för stora system kan hårdvara användas för att accelerera simuleringen. Simulink innehåller stöd för användning av Xtreme DSP kortet vid simulering. Detta innebär att Xtreme DSP kortet kan användas för att utföra den funktion som modellerats med Xilinx blockset.

Vid systemsimulering kan Xtreme DSP kortet användas tillsammans med en Simulinkmodell av det övriga systemet. Denna funktion kan utnyttjas vid integrering av FPGA funktionen i systemet då en simulering först kan genomföras för att öka chansen att integreringen lyckas.

9.3 Kommunikation mellan Word och Simulink

Genom att använda MATLAB, Simulink och System Generator tillsammans med Microsoft Word har ett exempel på en metod för att koppla ett dokument skrivet i Word till den modell som används för att skapa en realisation i en FPGA tagits fram.

För att skapa länken mellan ett dokument skrivet i Word och en modell i Simulink som använder System Generator för att generara en FPGA realisation används variabler i MATLABs globala arbetsyta. Länken illustreras i figur 28.



Figur 28: Koppling mellan Word och Simulink

Via de globala variablerna kan värden på viktiga parametrar i modellen påverkas från ett dokument skrivet i Word. Det är också möjligt att presentera resultatet från en simulering direkt i Word. Genom att använda kopplingen mellan Word och Simulink garanteras att dokumentet alltid innehåller information från den senaste versionen av modellen och ingen manuell uppdatering av dokumentet krävs vid uppdatering av modellen.

Kopplingen mellan Word och Simulinkmodellen ger två fördelar:

- Det är möjligt att koppla en del av en kravspecifikation till modellen vid konstruktion av modellen. Detta gör att förändringar av kraven direkt kan speglas i modellen. Det är också möjligt att variera kraven och snabbt simulera eller generera en FPGA realisation för att se resultatet.

- Den andra fördelen är vid beskrivning av modellen. Genom att koppla beskrivningen direkt till modellen kan en rapport som alltid beskriver den senaste versionen av modellen genereras.

För att skapa en koppling mellan Word och Simulink har inbyggda funktioner i MATLAB och Simulink använts tillsammans med funktioner som skrivits för att passa den valda metoden.

9.3.1 MATLAB notebook

Den funktion som gör det möjligt att utbyta information mellan Word och MATLAB heter notebook. Det är en inbyggd funktion i MATLAB. Genom denna funktion är det möjligt att skapa fält i Word dokumentet som skickas till MATLAB för utvärdering. Dessa fält kan t ex innehålla variabeldeklarationer eller funktionsanrop. Resultatet av ett anrop till MATLAB skickas tillbaka för presentation i Word.

När notebook används skapas en så kallad M-book. En M-book innehåller text, MATLAB kommandon och resultaten av utvärderingen av MATLAB kommandon. En M-book använder makron för att kommunikationen mellan Word och MATLAB. Dessa makron är inte synliga för användaren.

Genom att använda notebook kan de kommandon som krävs för att göra uppdateringen av modellen placeras i ett Word dokument. Ytterligare kommandon för att öppna modellen, simulera modellen och analysera resultatet kan också placeras i dokumentet. Genom att använda notebook kan alla parametrar styras från ett dokument i Word och resultatet av de förändrade parametrarna kan analyseras i samma dokument

9.3.2 Modellens systemparametrar

Styrningen av Simulinkmodellen från Word sker i form av systemparametrar till modellen. Dessa parametrar används för att påverka modellens egenskaper. Alla modellens parametrar kan samlas i en dialogruta i modellen för att användaren enkelt skall komma åt dem.

9.3.2.1 Parametrisering av modellens block

Simulink innehåller stöd för att använda parametrar för de olika blocken i modellen. Dessa parametrar kan var konstanter, variabler från MATLABs arbetsyta eller uttryck som beräknas innan simuleringen startar.

När modellen konstrueras och en av systemparametrarna används vid parametriseringen av ett block används den globala variabeln från MATLABs arbetsyta som innehåller värdet av en systemparameter. Genom att använda den globala variabeln kan värdet av systemparametern läsas överallt i modellen.

9.3.3 Globala och lokala variabler

Simulink innehåller stöd för att använda MATLAB variabler i de modeller som konstrueras. Anledningen till att globala variabler används är den dialogruta som används för att komma åt parametrarna i modellen. För att skapa dialogrutan används en mask till ett tomt subsystem. Parametrarnas värden sparas då som variabler som är lokala för det tomma subsystemet. För att kunna använda värdet på en parameter utanför subsystemet används globala variabler.

För att de lokala och globala variablerna skall hållas synkroniserade används två funktioner, update_ws och update_model som skrivits i arbetet. Update_ws (update_workspace) användes för att uppdatera den global motsvarigheten till en lokal variabel och anropas varje gång en lokal variabel ändras. Update_model används för att uppdatera den lokala variabeln och anropas varje gång den globala variabeln ändras.

Hela källkoden till funktionerna update_ws och update_model bifogas i bilaga 1 repektive bilaga 2.

9.3.4 Begränsningar

Det faktum att modellens egenskaper påverkas genom parametrar innebär vissa begränsningar för hur modellens egenskaper kan varieras. Genom att endast parametrar kan påverkas är modellens arkitektur låst.

Eftersom modellen dessutom används för att generera en realisation i FPGA kan detta leda till en realisation som inte är effektiv. Till exempel kan en FIR filter struktur vara optimal för en uppsättning koefficienter men en annan struktur krävs för en annan uppsättning koefficienter. Detta är ett exempel på hur styrning av modellen begränsas av att endast parametrisering används.

Det är viktigt att vara medveten om hur parametrarna kan påvekas utan att strukturen av modellen behöver förändras. Denna begränsning måste konstruktören vara medveten om vid konstruktion av modellen.

10 Resultat

Resultatet av arbetet presenteras i detta kapitel. De viktigaste för- och nackdelarna med Simulink och System Generator presenteras tillsammans med en diskussion om vad datorstödd implementation innebär vid delsystemkonstruktion vid Ericsson Microwave Systems.

10.1 Simulink och Xilinx System Generator for DSP

De två testfall som gjorts visar att Simulink kan användas för simulering av de digitala funktioner EMW utvecklar och att Xilinx System Generator är ett bra alternativ för datorstödd implementation av digitala funktioner i en Xilinx FPGA.

10.1.1 Effektiv implementation

Utnyttjandet av FPGA resurser motsvarar utnyttjandet för handskriven kod. Det gör att System Generator kan användas för att uppgradera redan befintliga konstruktioner. Det är också ett viktigt resultat eftersom en ökning av FPGA utnyttjandet medför att större och därmed dyrare FPGA:er krävs.

10.1.2 Hög abstraktionsnivå

Genom att System Generator höjer abstraktionsnivån för konstruktion av FPGA funktioner kan utvecklingstiden förkortas. Dessutom bidrar den ökade abstraktionsnivån till att konstruktionen blir mer lättöverskådlig och lättare att presentera.

En högre abstraktionsnivå innebär dessutom att de beslut som fattas på systemnivå får större betydelse för den slutliga realisationen. Detta gör att ett större konstruktionsansvar hamnar hos delsystemkonstruktören vid konstruktion av digitala funktioner.

10.1.3 Systemkonstruktion

Användning av Simulink som utvecklingsmiljö medför nya möjligheter vid systemkonstruktion. Genom att använda en modell av systemet genom hela konstruktionsprocessen kan flera fördelar vid utvecklingen från algoritm till realisation utnyttjas.

- Genom simulering av funktionen genom hela utvecklingen ökas chansen att en fungerande hårdvara konstrueras vid första försöket.

- En gemensam utvecklingsmiljö för alla inblandade underlättar kommunikation mellan medarbetare i projektet.

- Verktygen medför en ökad abstraktionsnivå i konstruktionsarbetet vilket medför kortare utvecklingstid.

10.1.4 Begränsningar

Vid systemkonstruktion kan simuleringstiden för modellen göra att systemsimuleringar inte går att genomföra i praktiken. På grund av att de modeller som används för att generera en implementation arbetar med en tidskala som motsvarar klocktakten för systemklockan till FPGA:n tar simuleringar som motsvarar en realtid i termer av sekunder mycket lång tid.

Simulink och System Generator avänds för att implementera den digitala funktion som realiseras i en FPGA i det slutliga systemet. Övriga delar av systemet kan modelleras för att verifiera funktionen hos FPGA:n men konstrueras på traditionell väg.

Xilinx System Generator använder CE signaler för att realisera flera klockdomäner. Denna metod har brister vid reset av systemet. Xilinx är medvetna om problemet och arbetar med en lösning. Enligt representanter för Xilinx skall en lösning på problemet ingå i version 6.2 av System Generator.

10.2 Xtreme DSP utvecklingskort

De två testfall som behandlats under arbetet visar att en färdig hårdvaruplattform som utvecklingskortet Xtreme DSP kan användas för att göra prototyper av de digitala funktioner som ingår i de produkter som Ericsson Microwave Systems producerar. Vid valet av hårdvaruplattform kan dock några punkter vara viktiga att tänka på.

- Möjligheterna att kommunicera med kortet via både analoga och digitala signaler är viktigt. Undersök vilka krav som den aktuella funktionen kräver.

- Vilken kapacitet har den eller de FPGA:er ingår i plattformen? Finns det ytterligare funktioner i FPGA:n som kan utnyttjas? T ex specialiserade multiplikatorer eller inbyggt minne.

- Vilka möjligheter finns för klockning av FPGA:n? Finns det flera klocksignaler? Extern klocka eller oscillator?

- Hur kan FPGA:n programmeras? Kan FPGA:n konfigureras från ett minne som ingår i plattformen?

10.3 System Generator vid delsystemkonstruktion

Genom att System Generator används för konstruktion av de digitala funktioner som implementeras i FPGA:er påverkas delsystemkonstruktörens arbete. De beslut som fattas på delsystemnivå får direkt genomslag i den slutliga implementationen.

Den traditonella arbetsmetoden innebär att en kravspecifikation skrivs och lämnas till en digitalkonstruktör. Denna kravspecifikation blir i och med användningen av System Generator överflödig för de delar av systemet som implementeras i FPGA:er. Istället för en kravspecifikation överlämnas en modell som kan realiseras i en FPGA till digitalkonstruktören vars uppgift är att utföra implementationen.

När VHDL används för att konstruera en digital funktion är den slutliga timingen för systemet inte känd förrän efter place and route. Detta gör att modellen kan behöva modifieras för att implementationen skall uppfylla de krav som ställs för korrekt funktion. I detta arbete krävs erfarenhet av VHDL och de verktyg som används för implementation av VHDL.

Exakt var i arbetet med modellen och generering av VHDL digitalkonstruktören kommer in kan variera. I vissa fall kan arbetet med att bryta ner modellen till Xilinx blockset lämnas över helt till digitalkonstruktören genom att en mer funktionell modell överlämnas från delsystemkonstruktören.

10.4 Sammanfattning

Arbetet visar att Simulink och Xilinx System Generator kan användas för att effektivt implementera en modell av en digital funktion direkt i en Xilinx FPGA. En förutsättning för att modellen och realiseringen skall uppvisa exakt samma funktion är dock att det problem med flera klockdomäner som System Generator uppvisar löses. En sådan lösning är under utveckling av Xilinx.

För implementering av den genererade VHDL representationen av modellen krävs erfarenhet av VHDL.

De exempel som gjorts under arbetet visar att färdiga hårdvaruplattformar som t ex Xtreme DSP utvecklingskort kan användas för att minska kostnaderna och utvecklingstiden för utvecklingen av demonstratorer och prototyper på Ericsson Microwave Systems.

Användning av verktyg för direkt implementation av en modell i hårdvara medför förändringar i den traditionella utvecklingsprocessen vid elektronikkonstruktion. Kravet på särskild dokumentation kan minskas om istället en modell av systemet tillåts vara del av det material som överlämnas mellan olika avdelningar i konstruktionsarbetet.

11 Referenser

[1] Computer Aided Design of Electronics, Zebo Peng
http://www.ida.liu.se/~TDTS01/lectures/03/lec1.pdf
Dec 2003

[2] Ericsson Microwave Systems
http://www.ericsson.com/microwave/index.shtml
Aug 2003

[3] The MathWorks: Developers of MATLAB and Simulink for Technical Computing, http://www.mathworks.com
Nov 2003

[4] MATLAB FAQ,
http://www.utexas.edu/math/MATLAB/Manual/faq.html, Nov 2003

[5] Using Simulink, MathWorks, Inc., 2003

[6] The MathWorks – Analog/Mixed-Signal
http://www.mathworks.com/products/dsp_comm/anamix.shtml
Nov 2003

[7] DSP/Communication Design – User Stories
http://www.mathworks.com/products/dsp_comm/includestory.shtml?file=userstory2358.jsp, Dec 2003

[8] Alfke Peter, The Future of Field-Programmable Gate Arrays,
http://lebwshop.home.cern.ch/lebwshop/LEB99_Book/Plenary/04_Alfke.pdf, Nov 2003

[9] Xilinx What Is Programmable Logic?
http://www.xilinx.com/company/about/programmable.html
Nov 2003

[10] Virtex II platform FPGA, Data Sheet
http://direct.xilinx.com/bvdocs/publications/ds031.pdf
Nov 2003

[11] Sjöholm, Lindh, VHDL för konstruktion, tredje upplagan, Studentlitteratur, 1999

[12] Logic Synthesis,
http://www-ee.eng.hawaii.edu/~msmith/ASICs/HTML/Book/CH12/CH12.htm, Nov 2003

[13] Xilinx System Generator for DSP, Reference Guide
http://www.xilinx.com/ipcenter/dsp/ref_guide.pdf
Nov 2003

[14] The MathWorks – Product Page
http://www.mathworks.com/products/dsp_comm/xilinx_addons.shtml, Dec 2003

[15] Xtreme DSP Development Kit Users Guide, Nallatech Document NT107-0132, Nallatech Limited, 2003

[16] AD6644 Datasheet, Analog Devices
http://www.analog.com/UploadedFiles/Data_Sheets/14656404AD6644_c.pdf, Dec 2003

[17] AD9772 Datasheet, Analog Devices
http://www.analog.com/UploadedFiles/Data_Sheets/35959142539914AD9772A_b.pdf, Dec 2003

[18] LFSR Reference
http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm, Nov 2003

[19] Söderkvist, Ahnell, *Tidsdiskreta Signaler och System, 2:a upplagan* 1994

[20] Betts, J. A., Signal Processing, Modulation and Noise, Hodder and Stoughton Educational, 1970

[21] Råde, Westergren, Mathematics Handbook for Science and Engineering, Fourth edition, Studentlitteratur, 1998

[22] En FPGA signalbehandlar snabbare än en DSP,
Anna Wennberg, Elektroniktidningen 2003-12-09

[23] FPGA System-On-Chip soft IP design: A reconfigurable DSP,www.vlsilab.polito.it/~molino/scientific/papers/mwscas2002_dsp.pdf, Dec 2003

12 Bilaga 1 – Källkod update_ws

```
function update_ws(PARAM_MASK, varargin)
% Uppdaterar de globala Matlab variablerna som är associerade
till en Simulinkmodell.
%
% PARAM_MASK är namnet på det subsystem i Simulinkmodellen
% som innehåller de lokala variabler som skall uppdatera
% motsvarande globala variabler
%
% Om ett namn på en av Simulink variablerna skickas som
% inargumnet till funktionen uppdateras bara den Matlab
% variabeln som motsvarar den Simulink variabeln. Om inte
% något inargument ges uppdateras alla variabler till
% Simulinkmodellen.
%
% Henrik Eriksson 2003-10-09

if not(length(find_system('Name',PARAM_MASK)))
% Om det angivna inargumentet PARAM_MASK inte motsvarar ett
% giltigt subsystem är anropet felaktigt
      error(['ERROR !! --- Det finns inget subsystem '
      PARAM_MASK]);
end

if length(varargin) >= 2
    % Antalet inargument är större än lika 3. Detta är inte
    % ett giltigt anrop
    error('ERROR !! --- Fel antal inargument!');
end

if length(varargin) == 1
    % Två inargument är givna.

    variabel_namn=strrep(varargin(1),'_simulink','');
    % Tar bort _simulink från namnet på den lokala variabeln

    variabel_namn=char(upper(variabel_namn));
    % Konverterar namnet på variabeln till stora bokstäver.

    avail_vars=who('global');
    % avail_vars är en cell array av strings som innehåller
    % alla globala variabler.

    if sum(strcmp(avail_vars,variabel_namn)) == 0
        % Det finns ingen variabel som motsvarar det givna
        % inargumentet. En sådan skapas:
        disp(['Initierar global variabel: ' variabel_namn]);

        evalc( ['global ' variabel_namn] );
        evalin('base', ['global ' variabel_namn]);
        %skapar en global variabel för att motsvara den lokala
        %och deklarerar den i både funktionens och den globala
        %arbetsytan
    end
```

```
    % Deklarerar variabeln variabel_namn som global för att
    %göra förändringar i den globala versionen av variabeln
    evalc( ['global ' variabel_namn] );

    % Tilldelar den globala variabeln värdet från den lokala
    % variabeln
    evalc([variabel_namn '= ' get_param([bdroot '/'
    PARAM_MASK], char(varargin(1)))]);

else % Ett inargument är givet till funktionen

    % Läser in alla lokala variabler till variabeln mask_vars
      mask_vars=get_param([bdroot '/' PARAM_MASK],
      'MaskWSVariables');

    % N är antalet mask variabler
    N = length(mask_vars);

    avail_vars=who('global');
    % avail_vars är en cell array av strings som innehåller
    % alla globala variabler.

    for var_index = 1:N


variabel_namn=strrep(mask_vars(var_index).Name,'_Simulink','')
;
        % Tar bort _simulink från namnet på mask variablerna

        variabel_namn=upper(variabel_namn);
        % Konverterar namnet på variabeln till stora
        % bokstäver.

        if sum(strcmp(avail_vars,variabel_namn)) == 0
        % Det finns ingen variabel som motsvarar det givna
        % inargumentet. En sådan kommer att skapas

            disp(['Initierar global variabel: '
            variabel_namn]);

            evalc( ['global ' variabel_namn] );
            evalin('base', ['global ' variabel_namn]);
            %skapar en global variabel för att motsvara den
            %lokala och deklarerar den i både funktionens och
            %den globala arbetsytan
        end

       evalc([variabel_namn '= ' get_param([bdroot '/'
       PARAM_MASK], mask_vars(var_index).Name)]);
       % Tilldelar värdet från den lokala variabeln till den
       % globala variabeln

    end
end
```

13 Bilaga 2 – Källkod update_model

```
function update_model(PARAM_MASK,varargin)
% Uppdaterar en Simulinkmodell med värden från de globala
% variabler som har lokala motsvarigheter i Simulinkmodellen.
%
% PARAM_MASK är namnet på det subsystem i Simulinkmodellen
% som innehåller de lokala variabler som skall updateras av de
% globala variablerna.
%
% Om ett namn på en av de globala variablerna skickas som
% inargumnet till funktionen uppdateras bara den variabelns
% lokala motsvarighet. Om inte något inargument ges uppdaters
% alla lokala variabler.
%
% Henrik Eriksson 2003-10-09

if not(length(find_system('Name',PARAM_MASK)))
    % Om det angivna inargumentet PARAM_MASK inte motsvarar
    % ett giltigt subsystem är anropet felaktigt.
    error(['ERROR !! --- Det finns inget subsystem '
    PARAM_MASK]);
end

if length(varargin) >= 2
    % Antalet inargument är större än lika 3. Detta är inte ett
    % giltigt anrop
    error('ERROR !! --- Fel antal inargument!');
end

if length(varargin) == 1
    % Två inargument är givna.

    variabel_namn = upper(char(varargin(1)));
    % Skapar en variabel med namnet på motsvarande variabel

    param_namn=strcat(varargin(1),'_simulink','');
    % Lägger till _simulink till slutet av variabel namnet

    param_namn=char(lower(param_namn));
    % Konverterar namnet på parametern till små bokstäver.

    param_struct=get_param([bdroot '/' PARAM_MASK],
    'MaskWSVariables');
    % param_struct är en struct som innehåller fält för var
    % och en av de lokala variablerna.

    param_exists = 0;

    for n = 1:length(param_struct)
        if strcmp(param_struct(n).Name, char(param_namn))
            param_exists = 1;
        end
    end;

    if param_exists == 0
```

```
            % Det finns ingen variabel som motsvarar det givna
            % inargumentet
            error(['ERROR !! --- Det finns ingen parameter '
            param_namn]);
        end

        % Deklarerar Matlab variabeln som global för att läsa den
        % globala variabeln.
        evalc( ['global ' variabel_namn] );

        % Tilldelar parametern värdet från Matlab variabeln
          set_param( [bdroot '/' PARAM_MASK] , param_namn ,
          num2str(eval(variabel_namn)));

    else % Ett inargument är givet till funktionen

        % Läser in alla lokala variabler till variablen
        % param_struct
        param_struct=get_param([bdroot '/' PARAM_MASK],
        'MaskWSVariables');

        % N är antalet lokala variabler
        N = length(param_struct);

        avail_vars=who('global');
        % avail_vars är en cell array av strings som innehåller
        % alla globala variabler.

        for index = 1:N

            param_namn=param_struct(index).Name;
            % en sträng som innehåller namnet på den aktuella
            % lokala variabeln

            variabel_namn=strrep(param_namn,'_simulink','');
            % Tar bort _simulink från namnet på den aktuella
            % lokala variabeln

            variabel_namn=upper(variabel_namn);
            % Konverterar namnet på variabeln till stora
            % bokstäver.

            % Deklarerar Matlab variabeln som global för att läsa
            % den globala variabeln.
            evalc( ['global ' variabel_namn] );

            % Tilldelar den lokala variabeln värdet från den
            % globala variabeln
            set_param( [bdroot '/' PARAM_MASK] , param_namn ,
            num2str(eval(variabel_namn)))

        end
    end
```