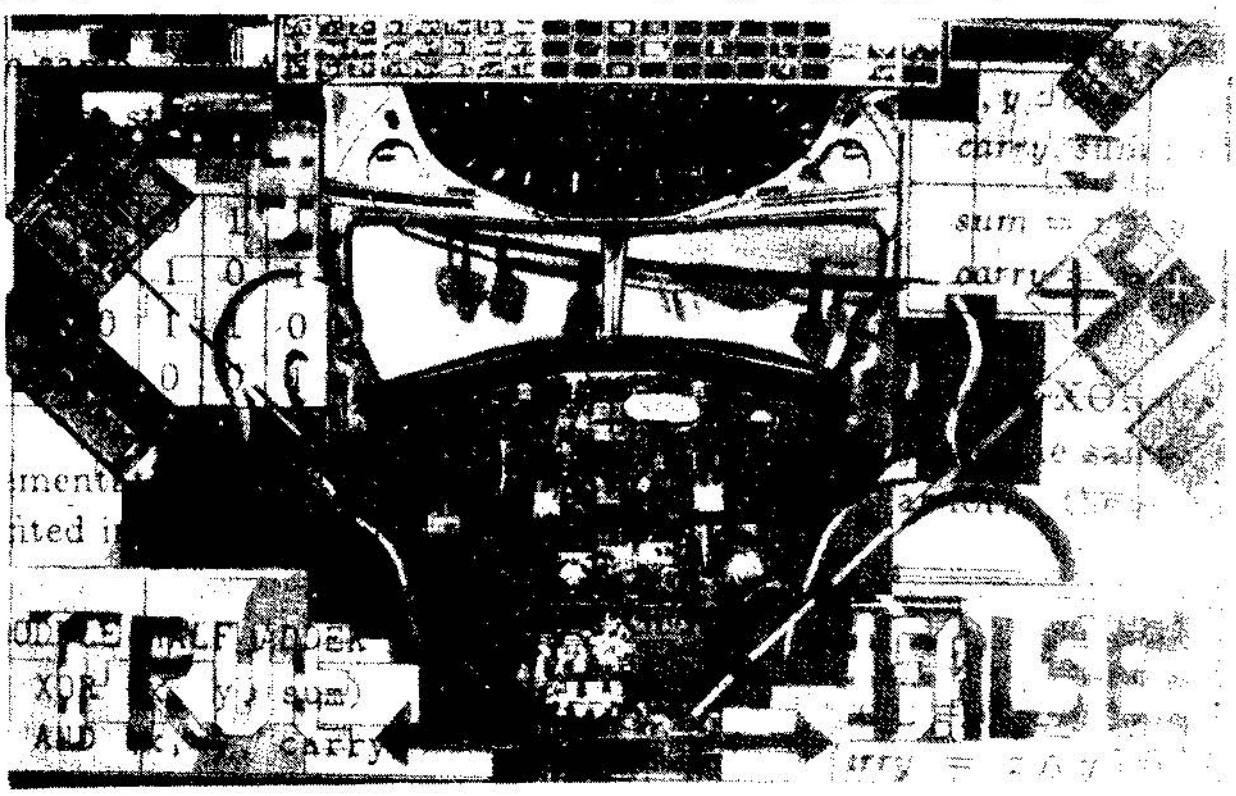
ÇALIŞAN YONGALAR ÜRETMEK*

Yazanlar: Davld SHEPHERD, Gr*g VVİLSON**
Çavlrl: Erhan YÜCEER'"

Kullandığınız bilgisayarın uygun olarak çalıştığını nereden biliyorsunuz? Programlarınız virüs taşıyabileceği gibi mikroişlemcinizin tasarımı da yanlış olabilir. Matematiksel mantık üzerine kurulmuş yeni bir yaklaşım güvenilmeye değer yongalar üretebilir.

Bir söylentiye göre bilgisayar mühendisleri uçağa binmezlermiş, çünkü uçağı kontrol eden elektronik yongaya güvenmiyorlarmış. Ortalama bir mikroişlemci, on veya yüz binlerce aktif bileşenden oluşan çok karmaşık bir aygıttır. Tasarımdaki en küçük hata yonganın hiç çalışmamasına, daha da kötüsü her bir iki milyon işlemde bir hata yapmasına yol açar. Bir mühendis bir yongayı kağıt üzerinde doğru olarak tasarlasa da tasarımın donanım olarak gerçekte uygulanması yanlış olabilir. Bir mimar bir köprüyü doğru olarak tasarlayabilir, fakat inşaatı yapan kişi ozaliti yanlış



(*) "Making Chips That Work", New Scientist. 13 Mayis 1989, sf. 61-64
(**) D. Shephard İnmos şirketi için yonga tasarlamaktadır. G. Wilson Edinburgh Üniversitesi'ndeki Edinburgh Parallel üstünbilgisayar (supercomputer) projesi'nde çalışmaktadır.
("*) ODTÜ Elektrik-Elektronik Müh. Bölümü 4. Sınıf öğrencisi

olarak tasarlayabilir, fakat inşaatı yapan kişi ozaliti yanlış okur ve kirişleri mimarın belirttiğinin yarı kalınlığında kutlanırsa köprü germe ve gerilme kuvvetleriyle çekebilir. Bu örneklere benzer olarak da, kağıttaki tasarımdan : jygu amaya hatalı olarak geçirilen bir yonga : ıı birasında yanılabilir.

Mikroişlemciler yaşam destek sistemleri, tren ve uçak kontrolları gibi bir çok uygulama alanları bulurlar ve bundan dolayı güvenilir olmak zorundadırlar. Hatta sıradan bir çamaşır makinasına bile yanlış bir parça koymak üreticinin çok para kaybetmesine yol açabilir. Günümüzde bazı yonga üreticileri matematiksel mantık kullanarak yonga tasarımında yeni tekniklere geçmişlerdir. Ağırlığını İngiltere ve ABD'nin oluşturduğu bir düzineye yakın araştır-ma grubu biçimsel yöntem adını verdikleri bu yöntemi geliştirmekedirler. Bu teknikler yongaları daha güvenli kılacaktır.

İngiliz yonga üreticisi olan Inmos firması ticari olarak yonga üretiminde biçimsel yöntemi kullanan Hk şirketlerden biridir. Şirket 1986 yılında serisinin üçüncüsü olan ve geleneksel işlemcilerden daha güçlü ve hızlı işlem yapabilen bir paralel işlemci olan T800 yongasını piyasaya çıkardı. Bu yonga çok büyük sayıların yer aldığı bilimsel hesaplamalar için idealdir.

T800 bir çok yongadan daha karmaşıktır. İçinde bir ana işlemci ünite, dört iletişim hattı ve dört kilobayt bellek vardır ve bunların hepsi bir çok tasarımda olduğu gibi ayrı değil bir tek yonga üzerindedir.

Ayrıca yongada kesirleri de içeren ve bilgisayarın çok büyük sayı gruplarını saklayıp işletebilmesine yardımcı olan bir ünite de yer almaktadır. Buna uçuşan nokta (floating poInt) aritmetiği adı verilmektedir. Herhangi bir sayıyı, bir tam sayının onun kuvvetleriyle çarpımı olarak gösterir. Örneğin 4096 sayısını 4.096×10^3 şeklinde göstermek olasıdır. Bilgisayar üs olan 3'ü 4.096'dan ayrı olarak saklar ve 4.096 sayısı ondalık noktası "uçuşan" olarak 4096 şeklinde saklanır. Ne yazık ki uçuşan noktalı sayılarla işlem yapmak daha karmaşık olduğundan tam sayılarla işlem yapmaktan daha yavaştır. Bilgisayardan 1.2×10^3 ve 3.3×10^{-1} gibi iki sayıyı toplaması istendiği zaman 4.5×10^2 cevabını almamak için bilgisayarın bazı kaydırma ve düzenlemeler yapması gerekmektedir. Büimsel hesaplamaların hızı bilgisayarın uçuşan nokta aritmetiğini başarma hızına bağlı olduğu için yonga tasarımcıları bu hesaplamalari hızlandırmak için özel uçuşan nokta aritmetiği üniteleri (FPU- floating point units) kullanmaktadırlar.

T-tarımcılar FPU'yu bilgisayarın diğer parçalarıyla aynı yonga üzerine koymakla bu yonganın kullanıldığı sistemnn genel hızını artırmayı umuyorlar. Yonga üzerindeki : ı-^J'nun içinde, yazmaç adı verilen ve çok hızlı olan belli.k hücreleri, uçuşan nokta aritmetiği yapacak olan e ve bu üniteleri birleştiren iletişim hatları (bus) vardır. Küçük ve değiştirilemeyen bir program olan mikrokod FPU'nun veri oluşturan rakamlara neler yaptığını, bu değerlerin hangi yoldan yazmaçlara, oradan bilgisayarın belleğine gittiğini ve geri döndüğünü kontrol eder.

Mikrokod FPUya gönderilen komutları mikrokomutlar düzeyine böler. Bu komutların her biri tek bir ünite veriyi bir yazmaçtan ötekine aktarır veya veriyi toplayıcı veya çarpıcı gibi aritmetik ünitelerden birine yollar. FPU'yu bu kadar karmaşık yapan şey, maksimum hız elde edebilmek için üreticilerin her bir mikrokomuta olası en çok işi yaptırma çabalarıdır. Bir tek komut, bir sayının bir parçasını bir yazmaca kaydırırken, ötekinin işaretini kontrol edip diğer bir üçüncü sayıyla toplama işlemine girebilir.

T800 gibi bir yonganın doğru tasarlanıp tasarlanmadığını görmenin en açık yolu onu yoğun biçimde denemektir. Yongaya olası tüm girdileri verip çıktıları inceleyerek teoride yonganın doğru çalıştığı söylenebilir. Örneğin toplamayı kontrol etmek için yongaya tüm olası sayı çiftleri girdi olarak verilip her bir çıktı incelenerek yonganın doğru çalıştığı söylenebilir. Fakat harcanması gereken çabanın büyüklüğü bu denemeyi olanaksız kılar.

Bunu görebilmek için bit'lik bilgiyi ele alalım. Bu bilgi iki durumda olabilir; 0 veya 1. iki bit birarada dört olası durumda bulunabilir: 00, 01, 10, 11. üç bit birarada sekiz olası durumda bulunabilir. Genelde, *n* bit kendi arasında 2^n değişik şekilde dizilebilir. T800 gibi birçok iç bellek bitlerine, artı, değişik giriş ve çıkış sinyal yollarına sahip bir mikroişlemcide ise tüm evrendeki parçacıklardan daha çok durum olabilir. Saniyede milyonlarca test yapılsa bile, bu yolla yonganın doğru çalıştığı kanıtlanamaz.

Sonuç olarak tasarımcılar yongayı tüm olası durumları temsil ettiğine inandıkları durumların küçük bir alt kümesiyle test ederler. Bu küçültülmüş alt küme bile, hala birkaç milyon test kapsıyor olabilir, bundan dolayı yongayı işletebilmek birkaç gün alabilir. Durumu daha da kötüleştirmek gerekirse, tasarım sırasında denemek için bir yonga bulunmadığından, yongaya benzeşen (simulate) bilgisayar pogramları kullanılmaktadır. Bu bilgisayarlar, tasarlanan yongadan yüzlerce hatta binlerce kez daha yavaş çalıştıkları için test zamanını aylara çıkarırlar. T800'ün taklit edilmesi sırasında, uçuşan nokta aritmetiğini test etmek için yapılan ve gerçeğin 1/500 hızında olan bir test neredeyse üç hafta sürmüştür.

Doğruluğu test ederek göstermenin doğuracağı en son sorun da test sonuçlarının değerlendirilmesidir. Milyonlarca test sonucunu tek tek incelemenin hiç bir pratik sonucu olmadığından, tasarımcılar yeni yongayı ya bilgisayardaki taklidiyle ya da piyasada bulunan başka bir yongayla karşılaştırmak zorundadırlar. Ama bu kez de tasarımcılar karşılaştırmada kullandıkları yonganın hatalı olabilmesi sorunuyla karşılaşabilirler. Inmos'da yapılan

ilk çalışmalarda böyle bir sorunla karşılaşılmış, referans olarak kullanılan yonga birkaç kez hatalı çıkmıştır. Inmos'un tasarım ve deneme yerine geliştirmekte olduğu yöntemlerle yonganın son halinin doğru olarak çalışması garanti altına alınacaktır. Bu yöntemler bilim adamlarının yüzeysel olarak benzemeyen iki terimin aslında aynı olduğunu göstermek için matematik eşitlikleri yeniden düzenlemelerine benzemektedir. Tasarımcılar yongaya yüksek düzeyli bir tanımlama yaparak işe başlarlar. Bu tanimlama her işlemden önce ve sonra hangi bileşen tarafından ve nasıl yapıldığını açıklamaksızm yonga bileşenlerinin durumlarını açıklar. Daha sonra tasarımcılar matematik teoremlerin açıklanmasına benzer yöntemler kullanarak bu tanımlamayı yonganın nasıl yapacağını ona çevirirler (translate). Bu anlatıma genellikle işlemsel anlatım adı verilir, çünkü matematiksel mantıkla oluşturulmuş ve hesap yapmada kullanılan bir dizi işlemden meydana gelmektedir.

Son olarak tasarımcılar bu işlemlerdeki komutları düşük düzeyli mikrokomutlar olarak, bir başka deyişle yonganın okuyabileceği bir bilgisayar koduna çevirirler. Bu yonganın her bir bileşenini kontrol edecektir. Çünkü bu işlemin her basamağındaki dönüşümler sadece tanımlamanın belirtildiği yolla değişir, yani ortaya çıkan mikrokod özgün tasarımın doğru bir çevirisi olmalıdır.

Inmos'taki tasarımcıların karşılaştıkları ilk sorun FPU yapısını oluşturduktan sonra, onu doğru çalıştıracak mikrokodun nasıl yazılacağı olmuştur. Başlangıç olarak tasarımcılar Amerikan. Elektrik ve Elektronik Mühendisliği Enstitü'sünden uçuşan nokta aritmetiği için bir standart aldılar. IEEE-754 olarak bilinen bu standart detaylı olarak her aritmetik işlemin sonucunun nasıl hesaplanacağını anlatmaktadır. Uygulayıcının lisansı için bir şey öngörülmemiştir, bir uygulama başarılı sonuç da verebilir, tersi de olabilir. Ne yazık ki IEEE bu standardı işe yaramayacak bir dil

Bölüm 1: Uçuşan Noktalar İçin Bir Mantık Testi

Geotf Barrett ve Inmos'taki tasarımcıların belirli bir komut için nasıl mikrokod geliştirdiklerini anlamak amacıyla, bir uçuşan nokta sayısının eşdeğer bir tam sayıya çevrilişi örneğini ele alalım. Bu iş genellikle daha önce ı saklanmış bir tam sayıyla toplama yapabilmek amacıyla yapılır. Bu çevirim yapılırken, öncelikle FPU uçuşan nokta sayısının değerinin bir tam sayiya çevrilip çevrilemeyeceğini test eder. Bunun yapılma nedeni saklanabilen uçuşan nokta sayılarının sınırının tam sayıların sınırından çok büyük olmasıdır.

Eğer uçuşan nokta sayısının değeri bilgisayarın saklayabileceği en büyük artı sayıdan büyük, ya da en küçük eksi sayıdan küçükse, çevirimi yapmak anlamsız bir sonuç ortaya çıkarır. Bu test "Test Integer Range" adı verilen bir komut tarafından yapılmıştır. Buna ait Z ta-nımlaması, yazmaç X ve hata belirleyicisi FLAG olarak FPU üzerine bir bellek hücresi şeklinde yazılır. Eğer X'te saklanan değer bilgisayarın tam sayı saklama sınırının dışındaysa, FLAG TRUE (doğru) vererek bir hata olduğunu gösterir. Tersi durumlarda FLAG değişmeden kalır. Z mantık gösteriminde bu olay aşağıdaki gibi gösterilir.

Test Integer Range
X,X': Ftoating Point Register
FLAG, FLAG: {TRUE, FALSE}

value (X) e (valid floating point representations of 32-bit integers}
X'=X
MinInt S value (X) ≤, MaxInt=> (FLAG'=FLAG)
(value'(X)<MinInt)or(MaxInt< value (X))=>(FLAG'=TRUE)

Bu tanımlamaya "şema" adı verilir. Şemanın üst yarısı kullanılan değişkenleri verir-burada X sayısı ve hata belirteci FLAG olduğu gibi. İlk satır X'in uçuşan nokta sayısı kapsayan bir yazmaç olduğunu belirtir ve FLAG sadece TRUE (doğru) veya FALSE (yanlış) durumlarında olabilir. (Gerçek Z tanımlamasında "valid floating point representations of 32-bit registers* (uçuşan nokta sayılarının geçerli 32-bit'lik gösterimi) deyimi bu düşünceyi daha iyi açıklayan matematiksel bir gösterimle verilmiştir.) Üs işareti işlemden sonra değişkenin değerini belirtir.

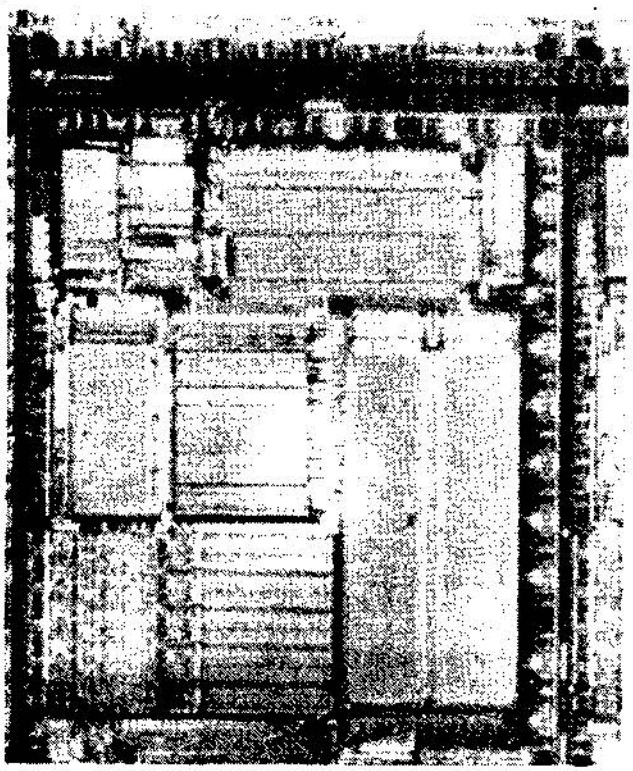
Şemanın alt yarısı değişkenlere bazı sınırlamalar getirir. Bu sınırlamalar işîamin etkilerini tanımlar. İlk satır işlemin geçerliliğim onaylamak için bir testtir-eğer X'in uçuşan nokta sayısı olarak değeri 32-bit'lik bir tam sayı değeri vermiyorsa hiç bir işlemin bir anlamı kalmaz. Bu komutu kullanan kişi yukarıdaki koşulu mutlaka sağlamalıdır.

Şemanın alt yarısındaki ikinci satır X'in değerinin işlemden sonra aynı kaldığını söylemektedir, başka bir deyişle X geçerli olmayan bir sayı olsa bile değişmemiştir. Bu işlem sırasında kazara sayının değişmesi yan etkilerden kurtulmak için tanımlamaya verilmiştir, tersi durumda olayları izlemek güçleşir.

Üçüncü satır, eğer X doğru sınırlar içindeyse, yani value (X) MinInt ve MaxInt arasında ise, FLAG değişmez ve bu önceki durumunu korur Tersi durumda, yani X doğru sınırların dışındaysa, dördüncü satır FLAG'in TRUE (doğru) sınırların dışındaysa, dördüncü satır FLAG'in TRUE (doğru) durumunu göstereceğini belirtir. Value () fonksiyonunun ve yazmacın 32-biflik geçerli bir sayı kapsadığının test edilmesi tüm FPU tanımlamasının bir parçasıdır.
FPU'nun yapması gerekenin yukandaki tanımlamalara uymak olduğu occam programlama dili kullanarak yazılan aşağıdaki işlem basamaklanyla kanıtlanabilir:

```
IF (MinInt<= value (X)) AND (value (X) <=MaxInt)      "eğer X'in değeri"
SKIP                                                  "tamamsa birşey yapma"
"mUE                                                  "fakat tüm diğer durumlarda"
FLAG:=TRUE                                            "FLAG1 TRUE (doğru) konumuna getir."
```

Testi değerlendirmek için aşağıdaki satıra ve bu basit programın FPU mikrokomutları cinsinden yazılmış biçimine gerek vardır.
(MinInt<= value (X)) AND (value (X) <= MaxInt)



olan İngilizce ile yazdı. Bir fizik probleminin çözümünün ilk aşamasının onu, bir dizi denkleme çevirmek olması gibi, FPU tasarımındaki ilk iş IEEE 754 standardının matematiksel mantığa çevrilmesi oldu. Oxford Programlama Araştırma Grubu'ndan Geoff Barett IEEE-754 standardını "Z" adıyla özel bir mantıksal gösterime çevirdi. Z gösteriminde her bir ayrı kural bir işlemin yapılması için sağlanması gereken koşullar ve FPU bileşenlerinin durumları cinsinden o işlemi tanımlar. Barrett, bundan sonra uçuşan nokta aritmetiğini IEEE tanımlamasına göre yürütecek bir program tasarlayabilmiş ve bu programın doğruluğunu kanıtlayabilmiştir. Barrett ve Inmoslaki tasarımcılar bir uçuşan nokta sayısını eşdeğer tam sayıya çevirme örneğinde olduğu gibi FPU komutlarına mikrokod geliştirmek için kullandılar.

Tasarımcılar IEEE-754 standartmdaki her işlem için bu şekilde Z tanımlamaları yazdılar. Bu tanımlamalar ne yapılması gerektiğini söylerken, nasıl yapılacağını belirtmezler. Mikrokod yazıcılarına daha sonra gereken şey ise, her Z tanımlamasını karşılayacak ve FPU'nun sürdürebileceği işlemler cinsinden yazılmış işlem basamakları oldu. Bu yazıcılar işlem basamaklarını yazmak için occam adı verilen basit bir bilgisayar dili kullandılar ve daha sonra geriye doğru çalışarak occam kodunun özgün Z tanımlamalarını sağladığını gösterdiler. Tasarımlardaki kanıtlamaları ekibin bir diğer üyesi David Shepherd tamamladı.

Occam'ın dönüşüm kurallarını kullanarak yüksek düzeyli işlem basamaklarını daha uzun ve FPU'nun mikrokod komutlarını kullanan düşük düzeyli işlem basamakları haline getirebilirsiniz. Bu işin içeriği sırasında bol kitap edinme gerekliliği ve hata yapma olasılığı vardır, örneğin Bölüm 1'deki dört satırlık program, her biri daha sonra bir düzine ya da daha fazla mikrokomutlara dönüşecek olan yarım sayfa dolusu bilgisayar komutuna dönüşebilir. Bunları kalem ve kağıtla izlemek ise tasarımcıların ilk başta biçimsel yöntem kullanarak kaçındıkları kadar hata yapılmasına yol açar. Fakat kitap edinmek oldukça mekanik bir işlemdir ve bir bilgisayar destekli tasarım (CAD) sistemi işin „oğunu halleder.

Bir kelime işlemcinin içerlek yazma, satır uzunluğu, sayfa biçemi gibi bilgileri aklında tutması gibi Oxford Programlama Araştırma Grubu tarafından geliştirilen bir CAD sistemi de program dönüşümlerini aklında tutar. Hangi kuralların kullanıldığını ve o ana kadar tanımlamanın ne kadarının çevrildiğini hatırlar. Bu sistem bir occam programını okur ve sonra programın bölümlerinde dönüşüm dizilerini uygular. Sistem programın o anki durumunu gösterir ve tasarımcıya hangi kuralların uygulanacağını soran bilgisayar daha sonra seçilen kuralı uygular, sonucu yazar ve tasarımcıyı tekrar uyarır. Tasarımcılar sık kullanılan kural kombinasyonlarına isim verebilir ve onları gerekli aritmetik teoremleri işleme koymadan kaydedebilirler.

FPU'nun ilk tasarımı dokuz ay almıştır. Geleneksel tekniklerle, eşdeğer bir ünite tasarlamak ise iki yıl sürmektedir. Tasarım ekibinde bir mimar, üç elektrik mühendisi, bir bilgisayar uzmanı ve iki matematikçi vardı. Biçimsel yöntem kullanılarak yazılan tüm mikrokodlar ilk seferinde çalıştı, fakat biçimsel yöntem kullanılmadan yazılan bazı FPU konutları çalışmadı. Buna ek olarak da ekip biçimsel yöntemi kullanmaya başlamadan önce FPU'nun donanım tasarımında bir çatlak meydana geldi.

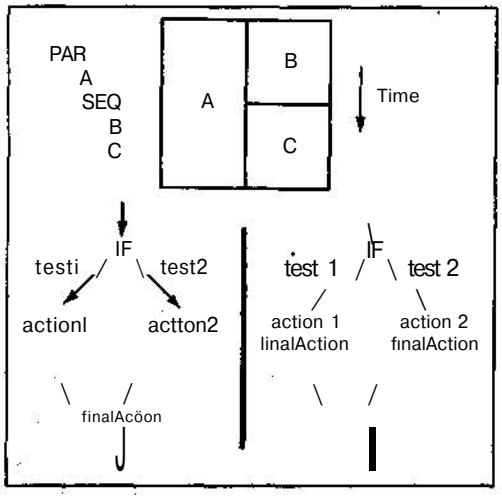
Yonganın ikinci kez gözden geçirilişinde, daha önce elle yazılan tüm FPU komutları bu kez biçimsel yöntem kullanarak yeniden yazıldı. Ne yazık ki bu yeni bir hata getirdi.Son anda elle yazılmış bir mikrokod değiştirilirken yanlışlıkla iki mikrokomutun adlarının yerlerini değiştirdiler. Sonuç olarak tüm yenileme boşa gitti. Çünkü böyle hatalar gelişme süresinden altı ile sekiz hafta götürürler ve böylece biçimsel metodun yonga tasarımının her tür işi için geliştirilmesi gerektiği ortaya çıktı.

Günümüzde Inmoslaki ekip İngiltere'deki tüm araştırma grupları gibi bu işle uğraşmakta. Inmos'un geliştirdiği ve "Şişman Freddie" olarak bilinen CAD sistemi, toplayıcılar gibi üniteleri, daha küçük ve basit altüniteier olarak yongaya tanımlayan hiyerarşik bir donanım tasarım dili (hardware design language-HDL) kullanmaktadır. Bu alt-üniteler ilerde mantıksal "VE" (and) ve "VEYA" (or) işlem çıktılarını veren en düşük düzeyli transistöre kadar bölünürler. (Mantık geçitleri iki sinyal alırlar: iki sinyal "açık" "TOPLA" (add); bir sinyal "açık" "VEYA" (or) demektir.) Bu şekilde çalışmak tasarımcıların sık kullanılan bileşenler hakkında daha sonraki tasarımlarda kullanmak üzere bir kitaplık oluşturmasını sağlar.

Bölüm 2: Occam ile Program Kanıtlanması

Araştırmacılar iki nedenden ötürü oceam'ı biçimsel yöntem programlama dili olarak seçtiler. İlk olarak, diğer dillerin aksine, occam'la birkaç işin birarada yapıldığı programlar yazmak olasıdır. Bu olay, eğer FPU'daki bazı yazmaçların ve aritmetik ünitelerin sıralı hareketlerinin program tarafından anlatımı içinse kesinlikle gereklidir.

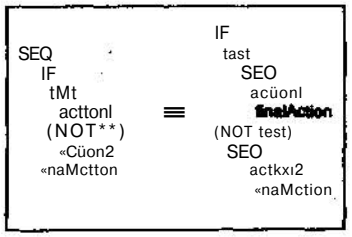
Örneğin occam'ın SEQ ve PAR adı verilen iki işlemi vardır. SEQ ar - dışık (sequentlal) anlamına gelir ve bir grup işlemin arka arkaya yapılacağını belirtir. PAR ise koşut (parıllel) anlamındadır ve bir grup işlemin aynı anda yapılacağını belirtir. Aşağıdaki tanımlama A,B ve C işlemlerinin Şekil Vdeki gibi yapılacağını anlatır.



(Occam'da içeriye yazarak SEQ işleminin B ve C Özerinde, PAR işleminin ise A ve ŞEQ üzerinde etkili olduğu belirtilir.)

İkinci olarak, occam'ın çok iyi tanımlanmış matematiksel özellikleri vardır ve bunlar occam programlarıyla kanıtlamayı Basic ve Fortran gibi dillerle yapılmış programlarla kanıtlamaktan daha kolay kılar. Bu kanıtlara ait kurallar, Edinburgh'tan Robin Milner ve Oxford'dan Tony Hoare'un teorik çalışmaları esas alınarak, yine Tony Hoare ve Bili Roscoe tarafından Oxford'da geliştiril* di.

Occam'ın kuralları, programlar arasındaki bir grup cebirsel eşitliktir. Cebirde ***ax(b+c)=(axb)+(axc)*** eşitlikleri olduğu gibi, occam programları işlemleri arasında da benzer eşitlikler vardır.



Bu eşitliklerin en basitlerinden biri de yukarıdaki dağılma özelliğine benzerlik gösterir. Bir programın, uygulanan test doğruysa action 1 tarafına, eğer test yanlışsa action 2 tarafına, ama sonuçta kesinlikle final action tarafına hareket ettiği bir sapak noktasını (branchlng polnt) ele alalım.

Final Action, eğer test doğruysa önce actionl daha sonra flnal-actıon veya eğer test yanlışsa önce actlon2 daha sonra flnal-actıon durumunu belirtmekle eşdeğerdir. Yukarıdaki kural, programların ortak noktalarında onların boyutunu küçültmek için kullanılır. Diğer kurallar ve kombinasyonları, programın yaptığı işi değiştirmeden yolunu değiştirmek açısından kullanıcıya yardımcı olur.

D

"HDL'nin doğruluğunu kanıtlamak hala bir sorundur. HDL bileşenlerin nasıl davrandıklarını değil ne şekilde bağlanacaklarını tanımlar. „

Inmos'un bugünkü CAD sistemi kuralları otomatik olarak kontrol edebilir. Her bileşenin tanımında o bileşenin nasıl kullanılacağını anlatan bir yönetmelik vardır. Örneğin bazı bileşenlerin sınırlarında üretim toleransları-için boş yer bırakmak gereklidir. Eğer tasarımcı bu tip bileşenleri çok yakına getirirse CAD sistemi tasarımcıyı uyarır ve tasarımı veritabanma eklemeyi reddeder. Buna ek olarak, tasarımcı bir modülün planını bitirdiği zaman, doğru bileşenlerin kullanılıp doğru bağlan-dıkları HDL tanımlamalarına göre kontrol edilir. Bu sistem tasarımcıların HDL sistemine uygun devreler üretmelerini sağlar.

HDL'nin doğruluğunu kanıtlamak hala bir sorundur. HDL bileşenlerin nasıl davrandıklarını değil ne şekilde bağlanacaklarını tanımlar. Günümüzde tasarımcılar bunu kontrol etmek için bilgisayar kullanmaktadırlar. Fakat Inmos'un T800 deneyiminde olduğu gibi bu da hataları önleme konusunda güvenilir olmaktan uzaktır. Inmos ekibi bugün her bileşenin özelliklerinin Z tanımlamalarını HDL kitaplığına eklemektedirler, Tasarımcılar bir kere yongayı planladılar mı, tanımlamaları kullanarak yonganın tasarımda beklenen özellikleri karşıladığı kanıtlanabilir. Sistem tasarımcıların donanım ve yazılım, yüksek düzeyle düşük düzey arasında hareket etmesine izin verir. Bu büyük ölçüde verimlürği artırır.

Bütün bunlar Kapalı bir kutu olan modern bilgisayar bilimini mühendislik disiplinine çevirmek için harcanan büyük çabanın bir parçasıdır; Kimse bir inşaat mühendisinden bir köprüyü ayakta durana kadar iki üç kez yapmasını beklemez. Biçimsel yöntemin yayılmasıyla, elektronik mühendisleri tarasımlarında aynı güvenilirliğe u^JşfiBytt<.ummakta, ürünlerini daha güv«B&)V$ rucuza üretmeye çalışmaktadırlar.^ ;6ııL