Széchenyi István Egyetem
Gépész Informatika és Villamosmérnöki Kar

Dr. Muka László

Dr. Lencse Gábor

Digitális rendszerek

Mikroprocesszoros rendszerek tervezése

Elektronikus jegyzet

v.0.91

2015

Tartalom

BEVEZETÉS MODUL

01 Rétegmodell – (ELSEVIER)

Az ábráról
A modell 9 réteget tartalmaz:

legalsó:	a fizikai réteg: itt elektronokról van szó,
következő:	eszközök: tranzisztorok, diódák, stb.,
következő:	analóg áramkörök: erősítők, szűrők, - elsősorban digitális jelfeldolgozásnál fordulhatnak elő,
következő:	digitális áramkörök: AND és NEM kapuk
következő:	logikai réteg: összeadók, memóriaegységek, stb.,
következő:	mikroarchitektúra réteg: adatutak és vezérlők,
következő:	architektúra: utasítások és regiszterek,
következő:	operációs rendszer: eszközmeghajtók/driverek
legfelső:	alkalmazói réteg: alkalmazói programok

Alkalmazói réteg tovább bomlik:
…
Digitális áramkörök rétege: Digitális hálózatok I. kurzus tartalma kiegészítve a logikai réteggel, amiben az összeadók és a memóriaelemek vannak: itt helyezkednek el a funkcionális elemek,
ebben a két rétegben megvalósíthatók a különböző automaták: a véges automaták, (Mealy és Moore automaták)

02 Rétegmodell (ELSEVIER)

A mikroarchitektúra rétegben helyezkednek el a programozható vezérlők, a mikroprocesszor, a mikrokontroller. A mikroszámítógép – a mikroprocesszor mellett – az alatta levő rétegekből is tartalmaz elemeket a mikroszámítógép-kártyára integráltan, míg a mikroprocesszoroknál és a mikrokontrollereknél az alatta levő rétegek a tokon belül fordulnak elő.
A tárgyalás folyamán találkozni fogunk az utasításrendszerrel és a regisztermodellel, aminek alapján tudunk dolgozni és programozni, és megnézzük az operációs rendszerek és a fordítóprogramok szolgáltatásait, és ezeket a továbbiakban a megfelelő helyen részletesen kifejtjük.
Érintjük az alkalmazói szoftver réteg is: az esettanulmány(ok) példáiban programot fejlesztünk, amely egyszerű alkalmazói programnak tekinthető.

01 blokk hiányzik

02 blokk rétegmodell

A mikroarchitektúra rétegben helyezkednek el a programozható vezérlők, a mikroprocesszor, a mikrokontroller. A mikroszámítógép – a mikroprocesszor mellett – az

alatta levő rétegekből is tartalmaz elemeket a mikroszámítógép-kártyára integráltan, míg a mikroprocesszoroknál és a mikrokontrollereknél az alatta levő rétegek a tokon belül fordulnak elő.
A tárgyalás folyamán találkozni fogunk az utasításrendszerrel és a regisztermodellel, aminek alapján tudunk dolgozni és programozni, és megnézzük az operációs rendszerek és a fordítóprogramok szolgáltatásait, és ezeket a továbbiakban a megfelelő helyen részletesen kifejtjük.
Érintjük az alkalmazói szoftver réteg is: az esettanulmány(ok) példáiban programot fejlesztünk, amely egyszerű alkalmazói programnak tekinthető.

03 A digitális rendszer tervezése funkcionális elemekkel és mikrogépekkel, illetve az ehhez használt fogalmak definiálása

A funkcionális elemet korábban érintettük: ezek a szokásos funkciók megvalósítására szolgáló elemek (felsorolás a Benesóczky féle könyvből): különösen fontos a programozható vezérlők kérdése, hiszen a digitális rendszerekben a rugalmas funkcionalitás eléréséhez programozással juthatunk. A programozható vezérlő olyan eszköz, ami erre szolgál.
A programozható vezérlők tervezése megelőzte a mikrogépes tervezést, azonban most is létezik, bizonyos előnyös tulajdonságai vannak.
A mikrogépek alapján történő digitális rendszertervezés viszont sok egyéb előnnyel rendelkezik, a funkciók könnyű megvalósíthatóságát, rugalmasságát és összetettségét és változtathatóságát is nagyobb mértékben teszi lehetővé.

04 A különböző mikrogépek meghatározása a digitális rendszerekben

A mikrogép a digitális rendszer programozható vezérlője: alapvetően egy számítógép, ami CPU-ból, memóriából és input-output eszközökből áll.

Mikroprocesszor egy CPU, számítógép memória és input-output egység nélkül, mindez egy tokban integrálva (?). Alapvetően az általános célú CPU-kat értjük ez alatt.
Magában a CPU-ban aritmetikai-logikai egységet, tehát ALU-t, vezérlőegységet és regisztereket találunk. Ehhez a kapcsolatokat lehetővé tevő síneket is hozzáértjük.

Mikrokontroller: egy tokban integrálva egy mikroprocesszor, egy kis memória és input-output eszközök, olyan mennyiségben és olyan fajták, amik az adott feladatkör megoldásához alkalmasak.

Mikroszámítógép az egy mikroprocesszor, (vagy mikrokontroller) plusz memória és input-output egységek *egy* kártyán integrálva.

A nagy teljesítményű mikroprocesszorok az ALU-n, vezérlőn, regisztereken kívül CASH, tehát gyorsítótárakat is tartalmaznak

Megjegyzendő, hogy ma, a nagy teljesítményű szuperszámítógépek is mikroprocesszorokra épülnek – ezek nagy teljesítményű un. szupermikroprocesszorok.

041 Különbség a mikroprocesszor, a mikrokontroller és a mikroszámítógép között

A mikroprocesszor olyan mikroszámítógép, amiben nincsen memória (a mikroszámítógépben természetesen van). A mikroprocesszor megvalósulhat és általában egy chip-es rendszerként használjuk

A mikrokontroller olyan mikroszámítógép, amiben csökkentett mennyiségű memóriát találunk.

05 Aktuális áttekintés a legfontosabb mikroprocesszorokról és mikrokontrollerekről

Ide kapcsolódik az Intel aktuális elemlistája, hiszen a mai mikroprocesszoros rendszerek mindegyike digitális rendszer, amik ajánlott vagy szabványos elemkészlettel rendelkeznek, amik a rendszerépítéshez szükségesek.

Adjuk a mikrokontrollerek ugyanilyen áttekintését,
illetve néhány általunk fontosnak tartott, valamint a későbbiekben használandó mikroszámítógép jellemzőit is áttekintjük

06 Elméleti áttekintés a számítógépről

A digitális számítógép egy digitális adatfeldolgozó eszköz.

Alapvetően fontos a számítógép architektúrája.

Architektúra: a számítógép funkcionális egységei, azok kapcsolatai és utasításrendszere

A számítógépek közül számunkra fontos a Neumann-típusú számítógép, illetve a Neumann-elvek.
A Neumann-elveket külön részletesen át fogjuk tekinteni:

A Neumann-elvek szerint épített számítógépekben az adatok és a programok a számítógép saját memóriájában vannak tárolva, és nincsenek megkülönböztetve: az adatok és az utasítások azonos módon kezelődnek.
A Neumann-elvű számítógép 2-es számrendszert használ, ami szerencsés választás egyrészt a könnyű megvalósíthatóság miatt, másrészt a logikai és aritmetikai műveletek közötti átjárhatóság is ebben az esetben a két érték alapján egyszerű.

Minden Neumann típusú számítógép alapvetően egy Turing-gép: minden olyan feladatot meg tud oldani az alkalmas utasításkészletével, ami algoritmikusan megoldható. Ez azt jelenti, hogy van egy olyan véges lépésszámú, az adott utasításkészlettel leírt módszer (algoritmus), amivel ez megoldható. A Turing-gép ezeket a feladatokat meg tudja oldani, és a számítógép egy Turing-gépet valósít meg.

A Neumann-architektúra mellett számunkra fontos a Harvard-architektúra.
A Harvard-architektúra esetén kisléptékű párhuzamosítás történik a rendszerben: nevezetesen az adatokat és a utasításokat külön tárolókban tároljuk, tehát a hozzáférés

eszközeit is ennek megfelelően többszörözzük. Ennek számos előnyös tulajdonsága lehet. A mikrokontrollerekben nagyon elterjedt architektúra.
Amely eszközöket megvizsgálunk, azokban is Harvard-architektúrát használnak.
A Harvard-architektúrának vannak módosított változatai *(Modified Harvard Architecture*). A Harvard architektúra továbbfejlesztése a *SHARC (Super Harvard Architecture*). A későbbiekben erre is kitérünk.

Fontos megjegyezni, hogy a processzorok utasításkészlete lehet CICS és RISC alapján szervezett.

A Neumann-, más hivatkozások szerint Princeton-, és a Harvard felépítésű számítógépek elhelyezhetők egy mátrixban. Tanulságos megvizsgálni, hogy melyik mikrogép, mikroprocesszor illetve mikrokontroller a mátrixban hol helyezkedik el.

07 További fogalmak:

bit-slice processzor: vezérlése továbbfejleszthető a vezérlőszóhoz való további hozzáillesztés révén,
DSP (Digital Signal Processor) — digitális jelfeldolgozó processzor
mag---
multicore: több magos processzorok: ugyanazon a lapkán elhelyezett több CPU-t jelent
multithread :a *többszálúság*, ahol egy-egy feladaton belül/egy task-on belül szálakat különböztetünk meg, amik között a váltás nagyon gyors tud lenni.
GPU (Graphical Processing Unit) processzorok, amik elsősorban grafikai műveletek végrehajtására készültek, pl. a mátrix típusú műveleteket nagyon gyorsan tudják végrehajtani. Van egy olyan irányzat, ami szerint a GPU típusú mikroprocesszorokat más célokra is előnyösen lehet alkalmazni.

08 A beágyazott rendszerek meghatározása és példák a beágyazott rendszerekre

A digitális rendszerek illetve a mikroprocesszorok és mikrokontrollerek szempontjából beágyazott rendszerek azok a rendszerek, amelyekben a mikrogép mint rendszerfölépítő elemnek, alkatrésznek tekintendő.

beágyazott rendszerek: példák

09 1-es modul 09

A rendszertervezési módszerek és eszközök

A rendszertervezési módszerek és eszközök különböző megközelítésmódokat vagy paradigmákat használnak, ilyen például a hardver-szoftver coo-design, vagy a hardver-szoftver külön történő tervezése.
Ez más-más megközelítésmódot jelent, és így ezekhez más-más módszerek kapcsolódnak.
Például a coo-design esetén a szoftver technológiai és a hardver digitális rendszertervezési és technológiai kérdések együtt jelennek meg.

Ennek vannak hagyományos módszerei és eszközei, mint például az állapotgépek, az idődiagramok, a folyamatábrák, a rendszerfolyamati architekturális ábrák, stb. használata. Ehhez különböző eszközök is használhatók: emulátorok, szimulátorok különböző nyelvekhez és rendszerekhez.

10 - 1-es modul 10

A digitális mikrogépes rendszerek kapcsolata környezetével

Internetkapcsolat
Beágyazott rendszerek és a felhő (cloud) kapcsolata

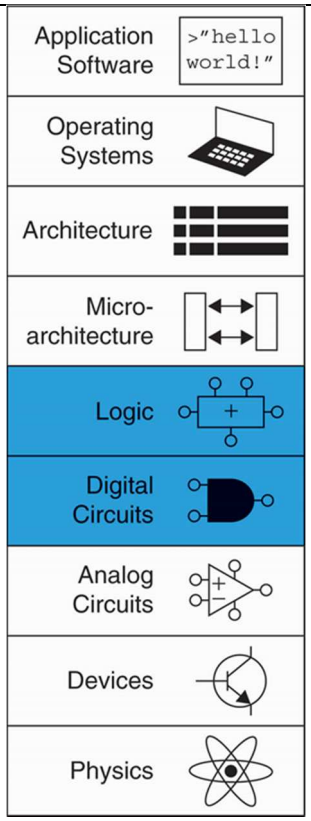
Rétegmodell

Digitális rendszer – réteg modell

forrás Elsevier

alkalmazói szoftver	programok
operációs rendszer	driver szoftver
architektúra	utasítások, regiszterek
mikro-architektúra	adatút vezérlők
logika	összeadók, memóriák
digitális áramkörök	NOT, AND kapuk
analóg áramkörök	erősítők, szűrők
alkatrészek	diódák, tranzisztorok
fizikai szint	elektronok

Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

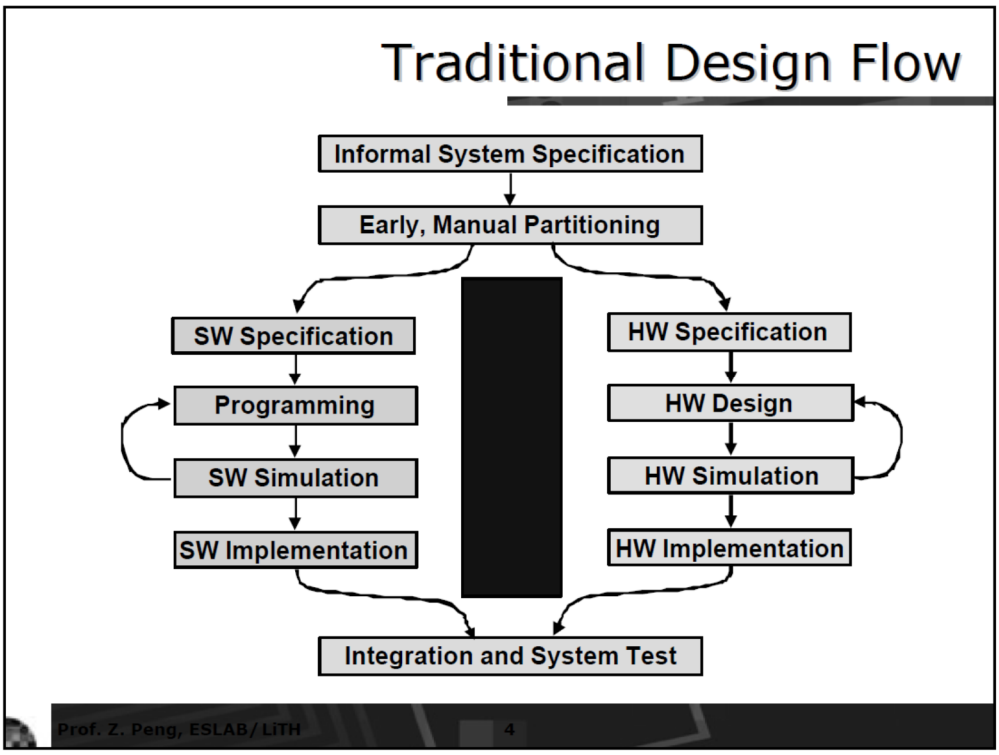


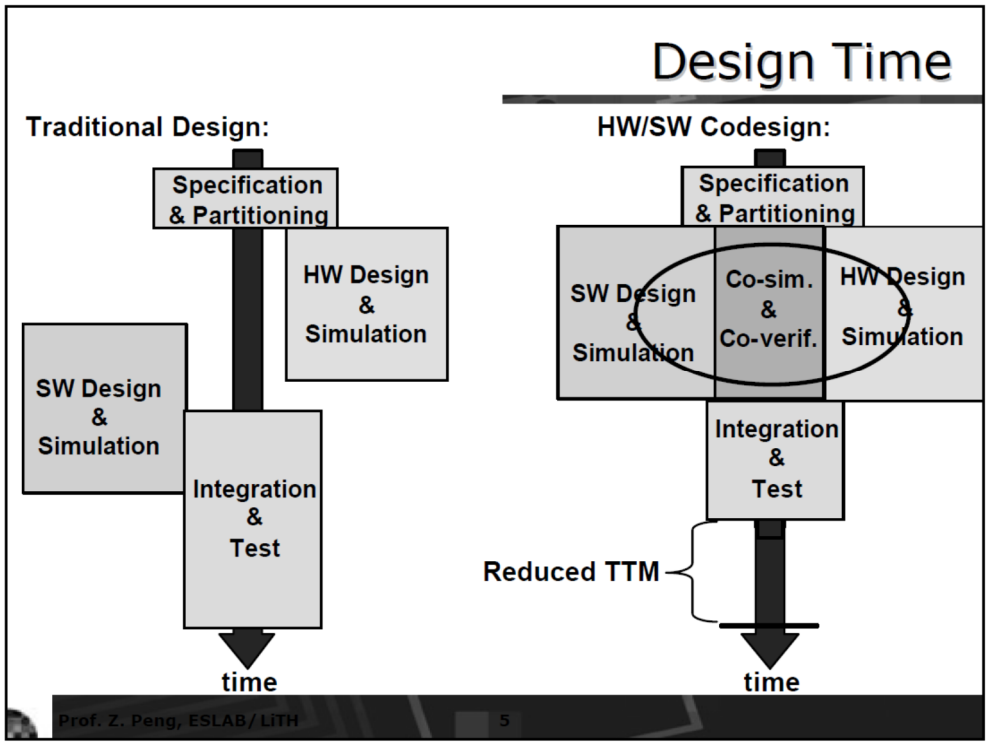
Forrás: saját (Tanenbaum, Elsevier)

10	alkalmazói szoftver	problémaorientált nyelv
9	assembly nyelvű program	assembly nyelv (mnemonik)
8	operációs rendszer	driver szoftver, ütemező, shell script
7	utasítás-rendszer architektúra	utasítások, regiszterek, ISA
6	mikro-architektúra	adatút vezérlők
5	digitális logika	összeadók, memóriák
4	digitális áramkörök	NOT, AND kapuk
3	analóg áramkörök	erősítők, szűrők
2	alkatrészek	diódák, tranzisztorok
1	fizikai szint	elektronok

Horizontális és vertikális codesign

Forrás: HW-SW Codesign tutorial-peng.pdf





HW/SW Codesign

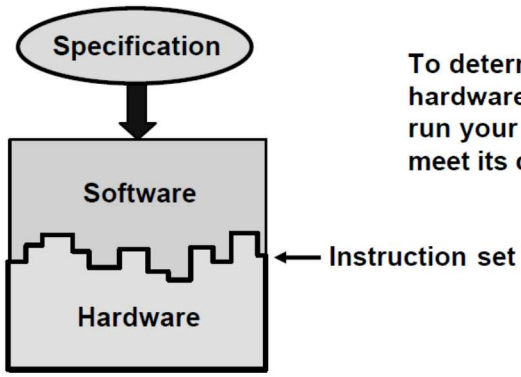
- The concurrent design of hardware and software elements, supporting explicit hardware/software trade-off.
  - Co-specification — to create an common specification that describes both hardware and software elements.
  - Co-synthesis — to concurrently synthesis the hardware and software implementations as well as their interfaces.
  - Co-simulation and co-verification — to simultaneously simulate and verify the hardware and software elements.

Prof. Z. Peng, ESLAB/LiTH 6

Vertikális codesign

Vertical Codesign

- Instruction set processor design, for both general-purpose systems and ASIPs (Application Specific Instruction Processors).



To determine how big the hardware engine you need to run your application and meet its constraints.

Prof. Z. Peng, ESLAB/LiTH 8

Codesign of Processors

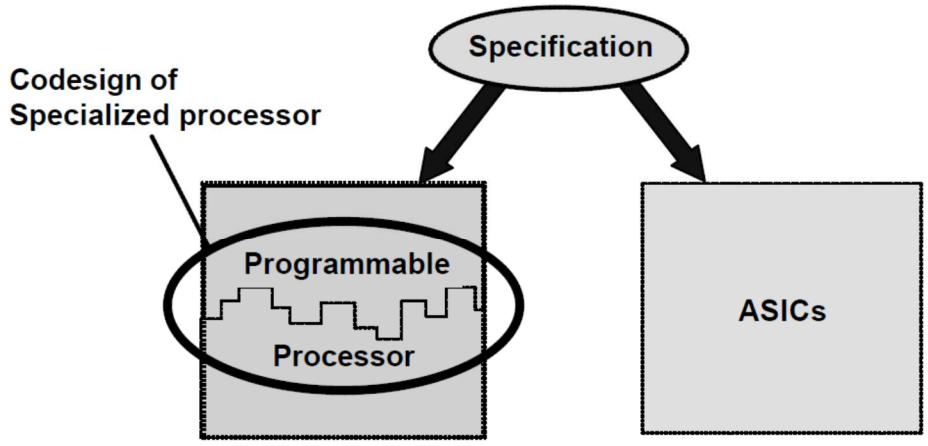
- General-Purpose Processors
  - Architectural support for operating systems.
  - Cache design and tuning (e.g., selection of cache size and control schemes).
  - Pipeline control design (control mechanisms, compiler design).
- ASIPs
  - Customization of instruction sets and specific resources (e.g., accelerator and coprocessor).
  - Design of register files, busses and interconnections.
  - Development of specific compiler.

Prof. Z. Peng, ESLAB/ LiTH 9

Horizontális codesign

Horizontal Codesign

- Some of system functionality is implemented in software running on programmable CPUs, while other functions are implemented in hardware.
- Typical for design of embedded systems.



Prof. Z. Peng, ESLAB/ LiTH 10

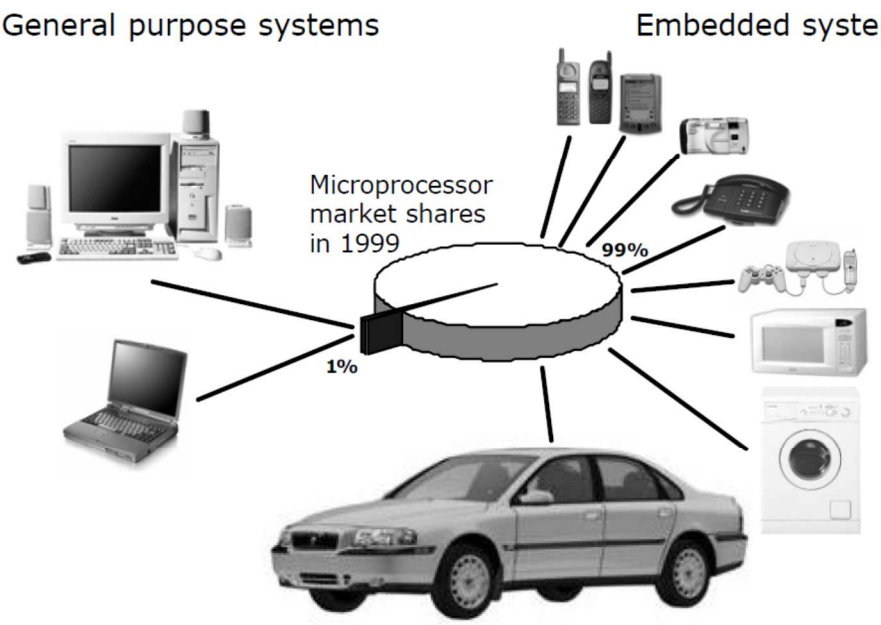
What is an Embedded System?

- There are many different definitions!
  - "A special-purpose computer system that is used for a particular task."
  - "A computer based systems embedded in real life machines. Though computer based, it dose not have the usual key-board and monitors. The processor and related circuitry are configured to do a specific task."
- Some highlights what it is (not) used for:
  - "Any device which includes a programmable component but itself is not intended to be a general purpose computer."
- Some focus on what it is built from:
  - "A collection of programmable parts surrounded by ASICs and other standard components, that interact continuously with an environment through sensors and actuators."

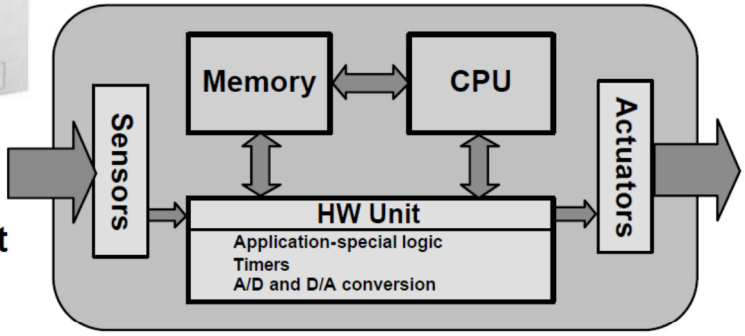
Characteristics of an Embedded System

- Dedicated (not general purpose).
  - One or several applications known at design-time.
- Contains a programmable component.
  - But usually not programmable by the end-user.
- Interacts (continuously) with the environment:
  - Real-time behavior.
  - Predictable.
  - Safe and reliable.
  - Run-time environment is fixed (faster ? better).
- Usually very cost sensitive:
  - Mass products in highly competitive markets and have to be shipped at a low cost.
- Low power is often preferred.

Embedded Systems

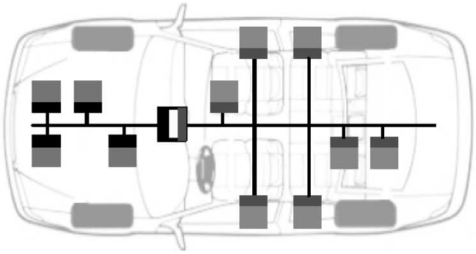


Embedded Controllers



- Reactive systems.
  - The system never stops.
  - The system responds to signals produced by the environment.

Distributed Embedded Systems



FÜGGELÉK

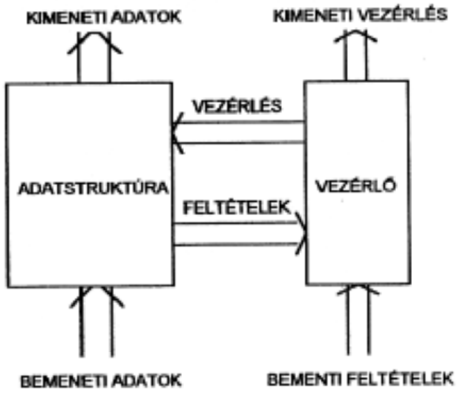
Adatstruktúra és vezérlő

Forrás: Benesóczky

3. TERVEZÉS ADATSTRUKTÚRA-VEZÉRLŐ SZEMLÉLETTEL

A megismert funkcionális elemek felhasználásával már viszonylag nagyobb áramkörök megtervezhetők, akár ad'hoch módszerrel is. Azonban egy terv annál kézbentarthatóbb, a tervezés annál több mozzanata gépesíthető, minél több szisztematikus módszert alkalmazunk a tervezés során. A címben szereplő eljárás is ezt célozza.

Alapötlete, hogy próbáljuk szétszedni két fő részre a feladatot, az egyik részt - ezt nevezzük adatstruktúrának - még mindig intuitív módon kell majd terveznünk, viszont a másiknál - ez a vezérlő - meg van a lehetőségünk a szisztematikus tervezésre, mert arra kidolgozott módszerek állnak rendelkezésünkre. Nagyon nagy méretű feladat esetén célszerű több egymástól jól elkülöníthető részre osztani azt, s egy-egy részfeladatra alkalmazni az adatstruktúra-vezérlő szemléletű tervezést.



3.1. ábra. Adatstruktúra-vezérlő felosztás

Ha egy feladatot a bemeneti és kimeneti adatok között elvégzendő, valamely algoritmussal megadott transzformációnak tekintünk, akkor az adatstruktúra a transzformáció egyes lépéseinek megvalósításához szükséges funkcionális elemek, és az azokat összekapcsoló adatutak összessége.

Az adatstruktúra rendelkezik egy felülettel a vezérlő felé. A vezérlő feladata, hogy az adatstruktúrából és a külvilágból jövő feltételjelek figyelembe vételével a vezérlőjelek segítségével megvalósítsa az adatstruktúrán az algoritmust (3.1. ábra).

Egy feladat hatékony megoldása természetesen nagyon függ az algoritmus megválasztásától. A választott algoritmus viszont nagyrészt meghatározza az adatstruktúrát. Adott adatstruktúra esetén már megadható a vezérlőjének folyamatábrája

(mely vezérlőjelet mikor kell kiadni a különböző feltételek függvényében). Ennek ismeretében pedig megtervezhető a vezérlő.

Vezérlőtervezés

Vezérlők tervezése szisztematikus módszerrel

A vezérlő tulajdonképpen egy sorrendi hálózat, melynek bemenetei az adatstruktúrából és a környezetből jövő feltételjelek, kimenetei pedig a vezérlőjelek.

A vezérlő típusának kiválasztásához a fő szempontok, hogy az algoritmus milyen bonyolult (hány állapotot igényel), milyen a sebességigénye, mennyire rugalmas (egy esetleges módosítás mennyibe kerül). Ez utóbbi szempontból a ROM-ot tartalmazó vezérlők (mikroprogramozott és mikroprocesszoros vezérlő) rugalmasabbak, mint a többi ún. huzalozott logika.

Random logika

Ide tartozik a hagyományos aszinkron, szinkron sorrendi hálózat, és az ad'hoch vezérlő tervezés.

Hagyományos aszinkron sorrendi hálózat

Csak nagyon kevés állapotszám és nagyon nagy sebességigény esetén jöhet szóba. Tervezése nagyon nagy tapasztalatot igényel, működése nehezen kézbentartható, nehezen tesztelhető. Mindezek miatt lehetőleg kerüljük az alkalmazását.

Hagyományos szinkron sorrendi hálózat

Megvalósítására nagyon alkalmasak a PAL-ok, PLA-k. Ma már nagy sebességgel (100 MHz felett) működő példányok is léteznek. Tervezésüket tervező rendszerek segítik, melyek akár magas szintű nyelven leírt kódolt (esetleg kódolatlan) állapotgráfból képesek megtervezni az áramkört (pl. ABEL nyelv). Ezeket alkalmazva nagy állapotszámú SSH (szinkron sorrendi hálózat) is tervezhető, szemben a régebben alkalmazott kézi módszerrel (Karnaugh tábla), amikor pl. 16 állapot felett, és 2-nél több bemenet esetén már nagy nehézségekbe ütközött a tervezés.

Ad'hoch vezérlő tervezés

Egy gyakorlott tervező ad'hoch módszerrel is képes az adatstruktúra vezérlését megoldani. Az ilyen megoldásokban azonban sok a hibalehetőség, a tervezés folyamata lassú, s a terv mások számára kevéssé áttekinthető (a tervező helyettesítése problémás, ha változtatásra van szükség) és nehezen módosítható.

Szisztematikus vezérlők

A szisztematikus vezérlőkhöz a fázisregiszteres, mikroprogramozott és mikroprocesszoros vezérlő tartozik. A szisztematikus vezérlők hardver kialakítása előre meghatározott struktúrát követ, szemben a random logikával. Tervezésük egyszerűbb,

jobban kézben tartható. Sokkal bonyolultabb vezérlések valósíthatók meg, mint a random logika esetében.

Fázisregiszteres vezérlő

A fázisregiszteres vezérlőnél a hálózat állapotát az ún. fázisregiszter tárolja. A tervező feladata a folyamatábra alapján, a struktúra feladatfüggő részeinek (kombinációs hálózat) megtervezése. Itt lényeges a különbség a hagyományos sorrendi hálózathoz képest, hogy a struktúra egy része adott, és hogy nem állapotgráf, hanem folyamatábra alapján kell tervezni, ugyanis az utóbbinál csak a kétfelé ágazás megengedett. A fázisregiszteres vezérlő egyik típusa a számlálós vezérlő.

Mikroprogramozott vezérlő

A mikroprogramozott vezérlők esetében a vezérlő működését egy ROM-ban tárolt mikroprogram határozza meg. A feladat bizonyos mértékű módosítása esetén, csak a ROM tartalmát kell módosítani, így ez sokkal rugalmasabb, mint az előbbi huzalozott logikák és sebességben sem sokkal marad le tőlük.

Mikroprocesszoros vezérlő

A mikroprocesszoros vezérlő a legkomplexebb feladatok megoldására is alkalmas, ami egyrészt a mikroproceszor aritmetikai képességeinek, másrészt a kialakított struktúra rendkívüli rugalmasságának köszönhető. Sebessége viszont némileg elmarad a többi vezérlő sebességéhez képest.

SBC-alapú vezérlő

Az UML használata

Forrás:Thesis A Platform-Centric UML-XML-Enhanced HW-SW 2004_.pdf

2.4 Other Embedded Design Approaches using UML

The platform-based design concept [14] upon which the chip-in-a-day approach is based also spawns an inception of the proposal for the Embedded UML profile [133] whose objective is to merge real-time UML and HW/SW codesign together. Embedded UML coalesces various existing ideas currently used in real-time UML and HW/SW codesign practices. It is conceived by its initiators as a UML profile package which is “suitable for embedded real-time system specification, design, and verification [133].”

In a nutshell, Embedded UML models the system using a collection of reusable communicating blocks similar to ROOM’s capsule concept [134]. Interfaces and channels, that are the extensions of ROOM’s port and connector notations, are used for communication specification and refinement. Within these interfaces and channels, UML stereotypes can be defined for communication and synchronization services. Other UML models such

as Use Case and Sequence diagrams provide means for specifying testbenches and test scenarios, while a combination of well-defined State diagram semantics and Action semantics [133] serves as a driving force for code generation, optimization, and synthesis. In addition, the extended Deployment diagrams, called the Mapping diagrams, may be employed to model system platforms [14] and furnish the platform-dependent refinement paradigm for performance analysis, and optimized implementation generation.

Even though no detailed implementation of the Embedded UML is available at the time of this writing, a careful perusal over its proposal reveals a few interesting facts. The Embedded UML profile and the UML profile for Codesign Modeling Framework (see Chapter 5) bear some resemblance as per their underlying objectives — each of which attempts to furnish a means to model and to develop platform-based real-time embedded systems. Certain minute implementation details of the two profiles differ tentatively. While the Embedded UML resorts to the ROOM’s real-time modeling approach [134] and utilizes capsule-based reusable blocks as the basic design units, the Codesign Modeling Framework profile relies on the UML profile for Schedulability, Performance, and Time Specification [29] and perceives all design entities, including communication links and protocols, as reusable objects in the LPO. Yet another fundamental difference between the two profiles exists that possibly comes as a consequence of discrepancy in the viewpoint towards how each of them should be conceived. Just like Vissers [9], this thesis believes that semiconductor manufacturers will design systems, rather than system houses design processors. Hence, it conceives the UML profile for Codesign Modeling Framework such that, when employed by the proposed method and/or the system developer, it allows for easy malleability to new technologies. This is to contrast with the Embedded UML profile whose approach seems to come from the opposite direction where system

houses are the primary forces in the development process. Under such a circumstance, it is likely that new methods and new ways defined by the semiconductor sector to build systems could possibly prevent the Embedded UML from attaining its full effectiveness — mainly due to the generalization of the Embedded UML package. As a preliminary assessment, the approach embraced by this thesis should eventually be more robust in a long run as it is

designed to better adapt to technological changes, and to be a more specialized package without totally relinquishing generality.

Proposed as part of the Yamacraw Embedded Software (YES) effort at the Georgia Institute of Technology, the YES-UML [135] represents yet another UML extensions package that aims at furnishing the system-level unified representation for the development of today's embedded systems. The YES approach fosters the concept of extreme reuse, where the development process encompasses both the Application Engineering and the Domain Engineering arenas, and the efficiency gain results from the application of the economies of scope [136] that suggest the development of a family of products rather than a single product as traditionally practiced.

The YES-UML is perceived as a complete system integrating notations whose objective is to "address the multiple-language, multiple-analysis problem of embedded systems design by combining together levels of abstraction and heterogeneous conceptual models [137]." Owing to its unified representation capability, systems analysts can deal directly with the UML models at the front-end, whereas systems designers can utilize their conventional analysis tools seamlessly at the back-end through the XMI interface. The proposed YES-UML extensions encompass the following modeling capabilities for [135]:

• Behaviors of analog interfaces within the embedded specification,

• Real-time related behaviors within the specification,

• Early identification of size, weight, and power (SWAP) constraints,

• Hardware/software interfaces, and

• The development of an executable specification capable of expressing concurrency in the real-time system being described.

It is clear from this discussion that, like the Embedded UML, the YES-UML also bears a number of similarities to the Codesign Modeling Framework profile even though its intended development paradigm is larger in scope than that of the proposed platform

centric approach. Due to lack of implementation details, no useful comparison between the YES-UML and the Codesign Modeling Framework can be made at this time.

2.5 A Perspective on Collaboration with Non-Platform Approaches

Being a graphical language, UML works well for the proposed approach to help promote reuse at a high abstraction level — specifically at the platform level where

the system could be quickly assembled using pre-designed, pre-characterized platform components. The proposed approach normally benefits from such platform-component reuse, as well as the use/reuse of knowledge brought forth by the XML technologies, to expedite the overall SoC systems development process. Nevertheless, the utterly diverse requirements of today's SoC systems make it nearly impossible that a desirable platform component would always exist for the system developer when applying the proposed platform-centric SoC design method. As a result, this section discusses a possible collaboration of the platformcentric approach and other non-platform approaches that could spawn an efficient subprocess within the proposed approach, and satisfactorily address this very issue.

A programming language such as SpecC and SystemC is particularly well-suited for implementing the functionality of the UML models. The reasons are that (1) both of them are object-oriented, as is UML, which could result in a convenient mapping scheme between the model and the implementation, and vice versa, and (2) like UML, they can uniformly represent hardware and software in a single language. SystemC, in particular, permits a large repertoire of C/C++ reusable modules to be incorporated should need arise. As a result, SystemC manifests itself as a possible language of choice for the proposed approach for the implementation of the UML models.

Forrás - haszn ---- KÖNYV ## NAGYON JÓÓÓÓ EMBEDDED & SYSTEM DESIGN principles UML ben ------ ELSEVIER 2008 .pdf

FORMALISMS FOR SYSTEM DESIGN

As mentioned in the last section, we perform a number of different design tasks at different levels of abstraction throughout this book: creating requirements and specifications,architecting the system,designing code,and designing tests.It is often helpful to conceptualize these tasks in diagrams. Luckily,there is a visual language that can be used to capture all these design tasks:the Unified Modeling Language (UML) [Boo99,Pil05]. UML was designed to be useful at many levels of abstraction in the design process. UML is useful because it encourages design by successive refinement and progressively adding detail to the design,rather than rethinking the design at each new level of abstraction. UML is an object-oriented modeling language.We will see precisely what we mean by an object in just a moment, but object-oriented design emphasizes two concepts of importance: ■ It encourages the design to be described as a number of interacting objects, rather than a few large monolithic blocks of code. ■ At least some of those objects will correspond to real pieces of software or hardware in the system. We can also use UML to model the outside world that interacts with our system,in which case the objects may correspond to people or other machines.It is sometimes important to implement something we think of at a high level as a single object using several distinct pieces of code or to otherwise break up the object correspondence in the implementation

However,thinking of the design in terms of actual objects helps us understand the natural structure of the system. Object-oriented (often abbreviated OO) specification

can be seen in two complementary ways: ■ Object-oriented specification allows a system to be described in a way that closely models real-world objects and their interactions. ■ Object-oriented specification provides a basic set of primitives that can be used to describe systems with particular attributes, irrespective of the relationships of those systems'components to real-world objects. Both views are useful. At a minimum, object-oriented specification is a set of linguistic mechanisms. In many cases, it is useful to describe a system in terms of real-world analogs. However,performance,cost,and so on may dictate that we change the specification to be different in some ways from the real-world elements wearetryingtomodelandimplement.Inthiscase,theobject-orientedspecification mechanisms are still useful. What is the relationship between an object-oriented specification and an objectoriented programming language (such as C++ [Str97])? A specification language may not be executable. But both object-oriented specification and programming languages provide similar basic methods for structuring large systems. Unified Modeling Language (UML) — the acronym is the name is a large language,and covering all of it is beyond the scope of this book. In this section,we introduce only a few basic concepts. In later chapters, as we need a few more UMLconcepts,weintroducethemtothebasicmodelingelementsintroducedhere. Because UML is so rich, there are many graphical elements in a UML diagram. It is important to be careful to use the correct drawing to describe something — for instance, UML distinguishes between arrows with open and filled-in arrowheads, and solid and broken lines.As you become more familiar with the language,uses of the graphical primitives will become more natural to you. We also won't take a strict object-oriented approach. We may not always use objects for certain elements of a design — in some cases,such as when taking particular aspects of the implementation into account,it may make sense to use another design style. However,object-oriented design is widely applicable,and no designer can consider himself or herself design literate without understanding it.

1.3.1 Structural Description

ezt még meg kellene nézni!

Egyéb

Forrás: Benesóczky

2.7.6. FPGA fejlesztési környezet

Az FPGA-k tervezését CAD rendszerekkel végzik. Ezek a bonyolultabb feladatokból adódóan nagyobb bonyolultságú és nehezebben kezelhető szoftverek, a PLD tervező rendszerekhez képest.

Az FPGA tervezési ciklusa

a. A terv bevitele

A tervet kapcsolási rajzzal és/vagy hardver leíró nyelv segítségével lehet bevinni.

A kapcsolási rajz funkcionális makro blokkokból építkezhet, melyek lehetnek:

a. könyvtári funkcionális blokkok,
b. saját tervezésű funkcionális blokkok.

Kapcsolási rajz készítésekor többszintű hierarchia lehetséges (egy durvább kapcsolási rajz egyes részeit újabb kapcsolási rajzok definiálnak). A bevitt tervet a tervező rendszer ellenőrzi abból a szempontból, hogy teljesíti-e a különféle előírásokat (pl. szintaktikai szabályok).

b. Szimuláció és funkcionális verifikáció

A tervező rendszer létrehozza a bevitt terv belső logikai reprezentációját. Ezután a tesztvektorok alapján ellenőrzi, hogy a bevitt terv megfelel-e funkcionálisan a specifikációnak.

c. A terv leképzése a konkrét FPGA architektúrájára (technology mapping)

Ebben a lépésben a tervezős rendszer a logikai blokkok által biztosított elemek felhasználásával megtervezi a logikai függvényeket és tárolóelemeket.

d. Elhelyezés és huzalozás

A tervező rendszer a logikai blokkokat megfelelteti az IC-ben elhelyezkedő fizikai blokkoknak. Ezután megtervezi a fizikai összeköttetéseket (huzalozás).
Ez a legkritikusabb rész a terv minősége szempontjából. Érdemes inkább drágább, de jó minőségű tervező rendszert vásárolni.

e. A behuzalozott terv késleltetéseinek számítása

Itt a valódi áramköri késleltetéseket próbálja meghatározni a rendszer.

f. Szimuláció és időzítési verifikáció a számított késleltetéseket figyelembe véve

Időkritikus terveknél különösen nagyon fontos a valós időadatokon alapuló szimuláció (pl. setup time betartása), de itt derülhetnek ki az esetleges hazárdok is.

g. FPGA konfigurációs adatok létrehozása

A konfigurációs adatokat (esetleg szabványos file formátumban) létrehozzák és tárolják.

h. FPGA konfigurálása vagy programozása

A konfigurációs adatok alapján a programozó készülék felprogramozza az FPGA-t vagy az EPROM-ot (Xilinx).

i. A beprogramozott eszköz teszteléseA beprogramozott eszközt a valós működési környezet szimulálásával, ill. a valós környezetben tesztelik. A teszteléshez hozzátartoznak a parametrikus vizsgálatok is. Pl. ellenőrzik, hogy a hőmérsékleti, tápfeszültség stb. határokon hogyan viselkedik az eszköz.

Álvéletlen generátor, mint számláló

A szokásos bináris, decimális stb. számlálók helyett az FPGA struktúrához jobban illeszkedik, a már ismeretett, EXOR kapukkal visszacsatolt sh.f:regiszterrel megvalósitott számláló (álvéletlen generátor). Az elérhető állapotszám csak 1-el kevesebb, mint a bináris számlálóknál. A vezérlő függvények nagyon egyszerűek.

Hátránya, hogy az egymást követő állapotok kódja nem a megszokott növekvő bináris számok szerint következik, így az egyes kimenetek nem rendelkeznek állandó kitöltési tényezővel, ezért frekvencia osztóként nem alkalmazható ez a megoldás. Külső ROM címzésére is kényelmetlen, mert a beégetendő adatok címét is az álvéletlen generátorhoz kellene igazitani.

Az ilyen számláló modulusát törléses módszerrel nem lehet csökkenteni (a 000..0 állapotban bennt ragad), csak betöltéses módszerrel.