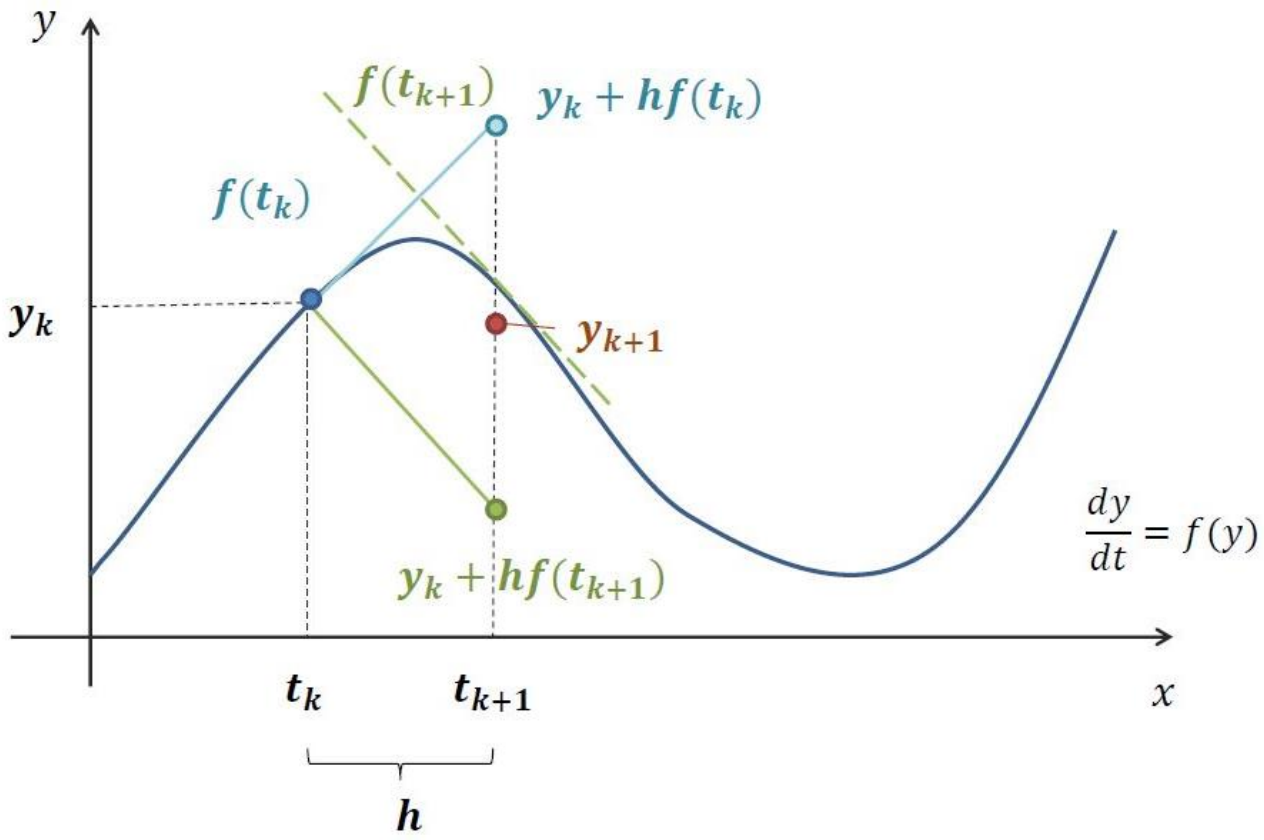
I. Szimuláció

1. Mi a változó lépésközű ODE solverek alapelve és miben jelentenek előnyt a fix lépésközűekhez képest? Milyen „trükkök” segítik a számításigény csökkentését?

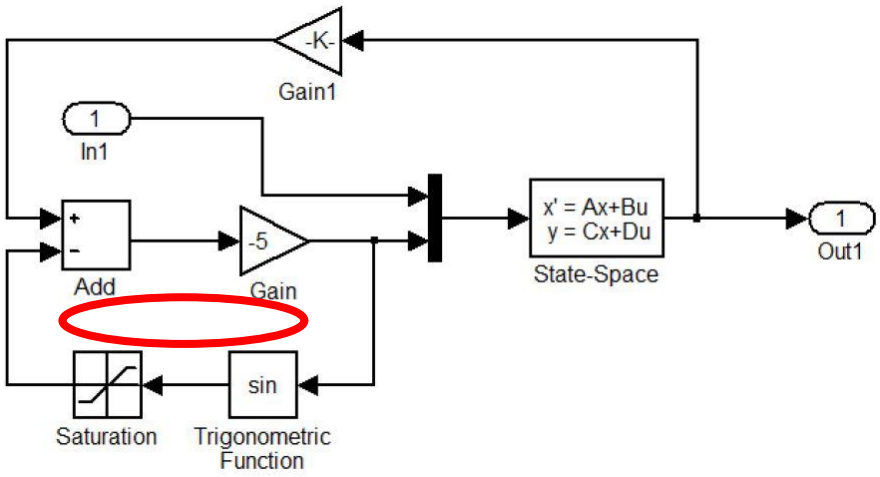
A változó lépésközű ODE solverek alapelve az, hogy az iterációs lépésközük nem fix, hanem a hiba függvényében dinamikusan változik. Azért előnyösek, mert ugyanolyan pontosság mellett jelentős mennyiségű számítást takaríthatunk meg velük.
Nullátmenet figyeléssel megspórolhatunk további számításokat, ha pl. Adott szakaszon belül N nullátmenetet detektáltunk, akkor úgy vesszük, hogy a függvény beállt nullába és vége.

2. Illusztrálja a Heun-módszer geometriai interpretációját egydimenziós esetben! Alkalmas ez a módszer stiff problémák szimulációjára? Adja meg a stiff rendszerek értelmezését!



Stiff rendszer alatt olyan dinamikus rendszert értünk, amiben nagyságrendben eltérő időállandók vannak jelen. A Heun módszer fix lépésközű, ezért nem alkalmazható stiff rendszerek megoldására, mivel azok változó lépésközt igényelnek.

3. Adja meg az algebrai hurok értelmezését! Milyen módszereket ismer az algebrai hurok feloldására? Jelölje meg az algebrai hurkot az alábbi blokk diagramon (az állapotteres blokk D mátrixa 0)!



Az algebrai hurok *direct feedthrough* elemekből álló hurok, amely mentén a blokkok be- és kimenetei egymástól körkörösen függenek.

Feloldása lehetséges:

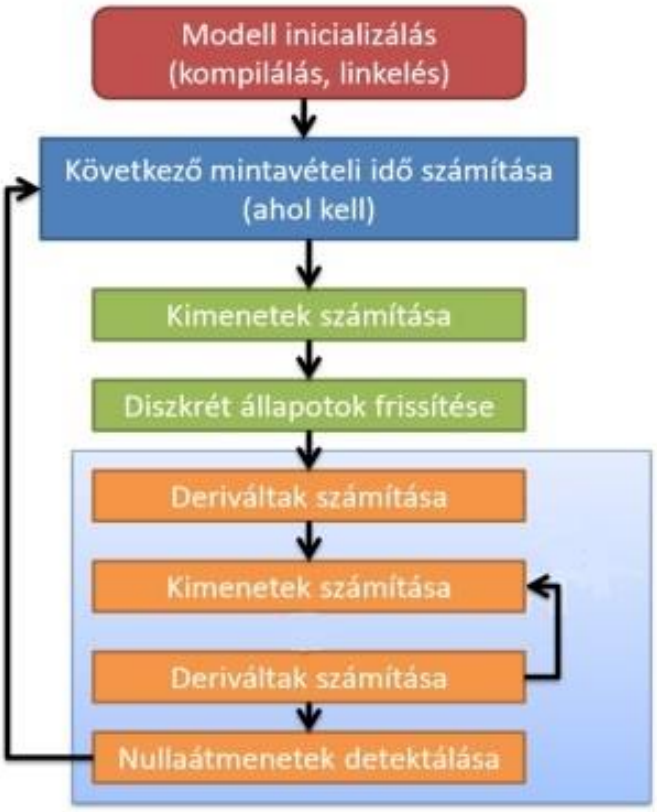
- Késleltetés beszúrása
- Az algebrai korlátozás differenciálegyenletté alakítása (nem mindig lehetséges)
- Koordináta transzformáció
- IC blokk beszúrása (kezdeti érték)

II. Simulink

1. Milyen lépéseket hajt végre a Matlab/Simulink egy modell szimulációjának indítása nyomán? Milyen metódusok meghívására kerül sor az egyes blokkok esetében az iterációs fázisban? Töltse ki az alábbi folyamatábrát, amely az iterálási szakasz lépéseit mutatja:

A START utáni lépések:

1. Modell kompilálás (compilation):
    1. Blokkok paramétereinek kiértékelése
    2. Jelek attribútumainak ellenőrzése
    3. Blokkok egyszerűsítése
    4. Virtuális alrendszerek behelyettesítése
    5. Blokkok sorrendbe rendezése
    6. Mintavételi idők regisztrálása
2. Linkelés (link):
    1. Memória allokáció
    2. Metódusok végrehajtási listái
    3. Modellépítő által megadott prioritási szintek figyelembe vétele
3. Iterálás (loop phase):



2. Mire szolgál az S-függvény? Milyen metódusok implementálása szükséges? Hogyan lehetséges C nyelvű kódok beépítése?

Az S-függvénnyel lehet egyenletrendszerekkel leírt, tetszőleges működésű blokkot megvalósítani. Akkor szoktuk alkalmazni, ha az összehuzalozás túl bonyolult vagy nehézkes.

Az alábbi négy metódus implementálása mindenképp szükséges:

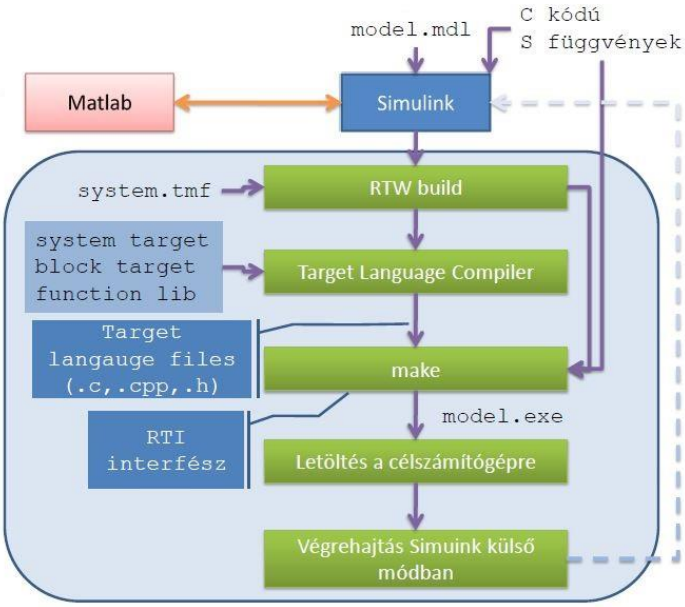
- Setup
- Derivatives
- Update
- Outouts

C nyelvű kód beépítése az S-function Builder blokk segítségével lehetséges.

3. Mit értünk egy Simulink modell „külső" (external) futtatásán? Milyen fordítási lépések szükségesek a külső futtatáshoz? Milyen korlátozásokat kell betartani egy Simulink modell esetében ahhoz, hogy azt „külső" módban futtathassuk? Magyarázza meg a Target fogalmát!

External futtatás során a szimuláció iterációs hurokban végzett számításait egy külső egység végzi el. Ennek a megvalósításához további VHLL → HLL → ExE fordítás szükséges. Továbbá két fontos korlátozás van a modellünk irányába: Nincs adaptív lépésköz választás, és valós idejű rendszer esetén nincs dinamikus memóriahasználat.
A Target az a számítógép / mikrokontroller, ahol a simulink modell futtatásra kerül.

4. A Simulink Coder a külső futtatáskor az alábbi ábrán szemléltetett lépéseket hajtja végre. Magyarázza meg, hogy mi történik az egyes lépések végrehajtásakor! Hogyan jelenik meg a célhardver kapcsolata a külvilággal a modellben?



<A lépések triviálisak ☺>
A célhardware kapcsolatát a külvilággal olyan jel - források és nyelők jelképezik a modellben, amit a célhardware gyártója bocsájt a rendelkezésünkre simulink modell könyvtár formájában.

III. Rapid Control Prototyping, "Hardware in the loop" és "Software in the loop" szimuláció

1. Oldja fel az RCP, HIL, SIL, RTI rövidítéseket, magyarázza el azok jelentését és sorolja fel a megvalósításukhoz szükséges szolgáltatásokat tartalmazó fejlesztő környezet hardver és szoftver elemeit, azok funkcióit! Nevezzen meg egy alkalmazási példát, ahol HIL szimulációra lehet szükség (indoklással)!

RCP: Rapid Controll Prototyoing, a gyors prototípus-tervezés angol megfelelője.
Szükséges hozzá egy VHLL programozási nyelven működő software, valamint egy ezt futtatni képes környezet.

HIL: Hardware In the Loop, olyan RT szimuláció, ahol a rendszer szenzorait, beavatkozó szerveit (vagy azok egy részét) elektronikusan emuláljuk.
Az RCP eszközökön kívül szükséges RT kapcsolat egy erre a célra megfelelő külső hardwarehez.

SIL: Software In the Loop, olyan RT szimuláció, ahol a rendszer szenzorait, beavatkozó szerveit (vagy azok egy részét) programozottan emuláljuk.
Ugyanaz kell hozzá mint a HIL-hez.

RTI: Real Time Interface, Egy olyan (grafikus) felület amin figyelemmel kísérhetjük a modellünk egyes részeinek működését.
Ugyanaz kell mint HIL-hez plusz még valamilyen GUI.

HIL szimuláció szükséges lehet ott, ahol adott esetek tesztelése a valódi irányított rendszerrel veszélyes lenne, vagy nem lenne reprodukálható.

IV. dSPACE és Simulink

1. Osztályozza a dSPACE által gyártott hardvereket és soroljon fel 4 I/O típust, amely ezeknél a hardvereknél hozzáférhető!

A dSPACE eszközöknek két fő csoportja:
- Egy-kártyás eszközök
- Moduláris (többkártyás) eszközök

Jellegzetes I/O portok:
- TCP/IP
- USB
- FlexRay,
- CAN
- ADC/DAC

2. Sorolja fel, hogy milyen taszk típusokat kezel a dSPACE beágyazott operációs rendszere! Milyen állapotokban lehetnek ezek a taszkok és milyen elvek szerint történik az ütemezésük?

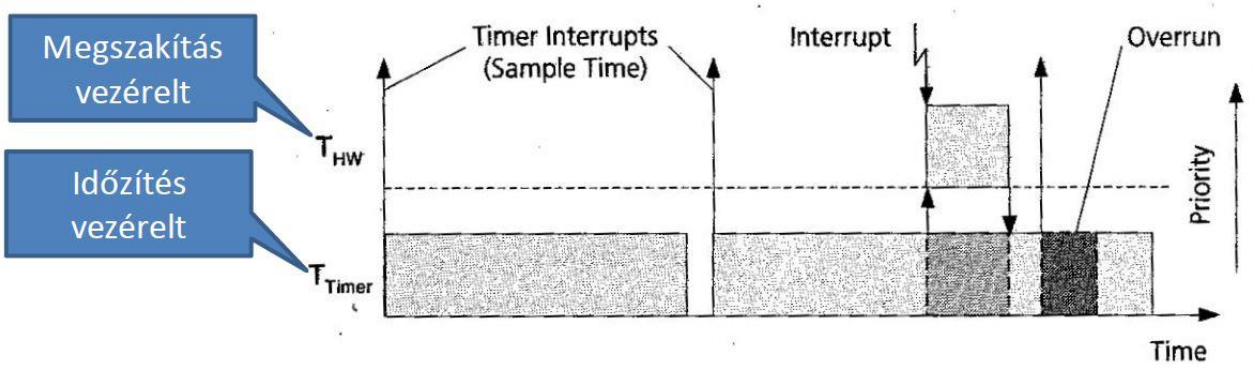
Három taszk típust különböztet meg a dSPACE operációs rendszer:
- Időzítővel vezérelt (Timer task):
  - Mintavételenként lefut
  - Az időzítő a processzor kártyán levő timer
  - Ha másképpen nem rendelkezünk, akkor alapból minden taszk Timer task
- Megszakítás vezérelt:
  - A megszakítást generáló blokk vezérli
  - Aperiodikus a végrehajtású
- Háttér taszk:
  - Adminisztráció
  - Log
  - stb…

Taszk állapotok:
- Inaktív
- Futásra kész
- Fut

Ütemezési alapelvek:
- Ha futásra kész állapotba került egy taszk, és a futónál magasabb a prioritású, akkor elveszi a processzort az éppen futó taszktól.
- Ha nincs futásra kész magasabb prioritású taszk, akkor az alacsonyabb prioritású visszakapja a processzort.
- Azonos prioritású taszkok között FCFS ütemezés van.
- Ha nincs futásra kész taszk, akkor a háttértaszk fut.

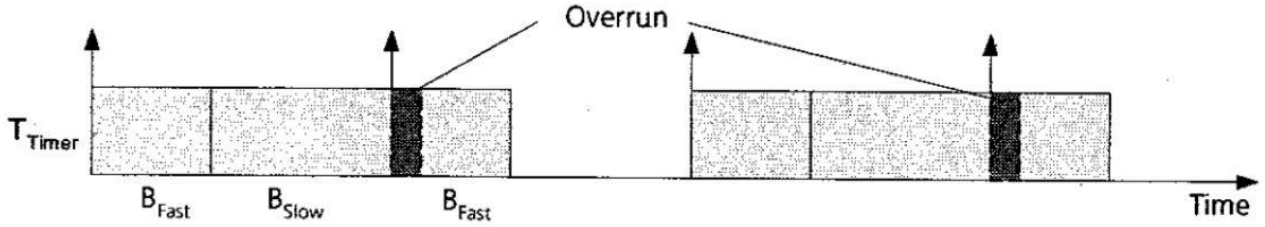
3. Magyarázza meg a ráfutás (overrun) jelenségét egy ábra segítségével akkor, ha a ráfutást egy megszakításvezérelt taszk okozza, illetve akkor ha „Single Timer” módban futtatunk különböző mintavételi periódusidejű blokkokból álló Simulink modelleket!

Megszakításvezérelt taszk:



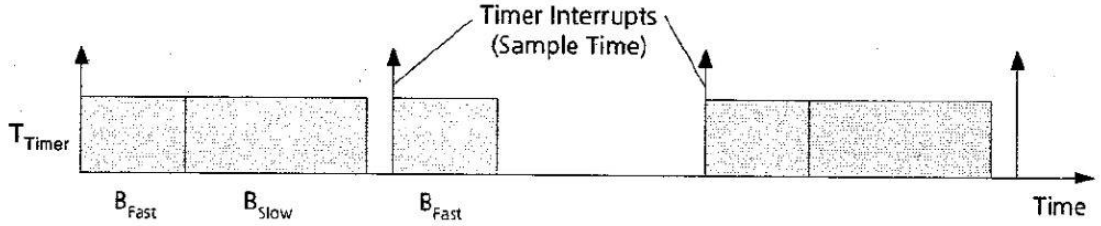
Ha egy taszkot úgy akarunk újraindítani, hogy az még fut, akkor úgynevezett ráfutás keletkezik.

Single Timer:

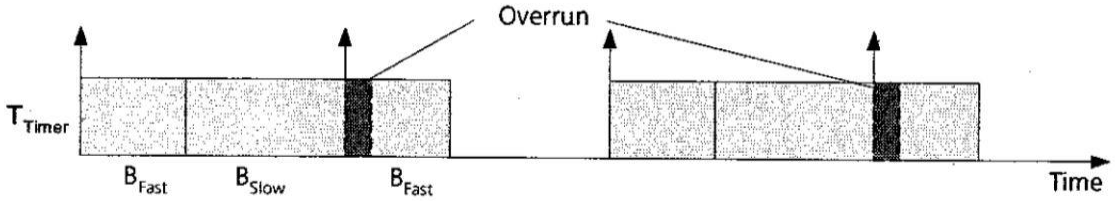


4. Mi a különbség a Single Timer és Multiple Timer ütemezés között? Hogyan lehetséges kiküszöbölni a ráfutás jelenségét Multiple Timer módban? (A megoldást szemléltesse idődiagramon!)

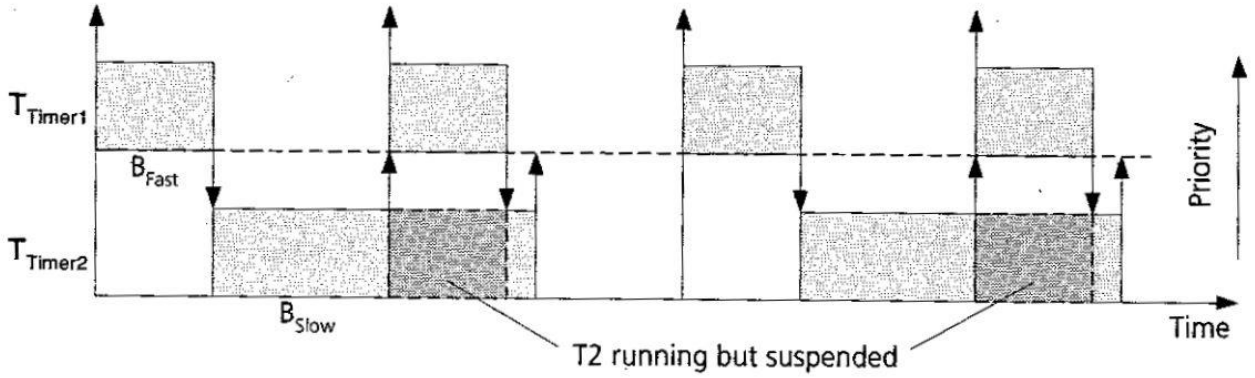
Megfelelően működő Single Timer ütemezés:



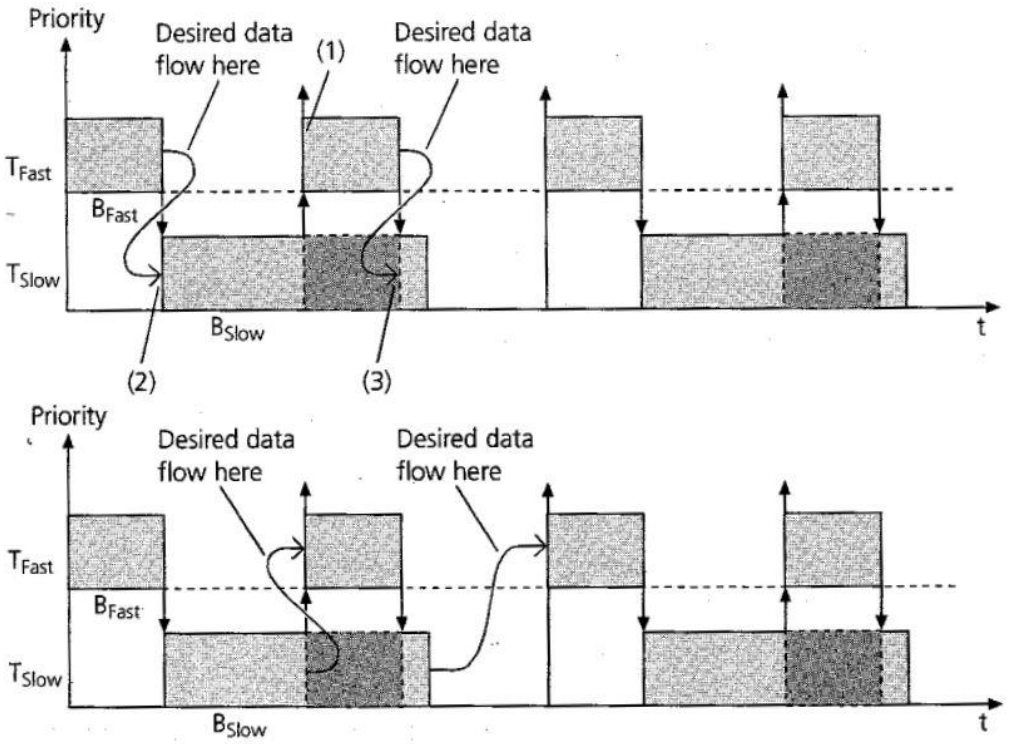
Lehetséges ráfutás Single Timer megoldás esetén:



Multiple Timer ütemezés során a hosszabb futásidejű taszk prioritása alacsonyabb, így kiküszöbölhető a lehetséges ráfutás:



5. Milyen adatkonzisztencia problémák adódhatnak, ha Multiple Timer ütemezés mellett egy lassú és egy gyors időzítéssel működő taszk egymás eredményeit felhasználja? (Mindkét lehetséges sorrendet vizsgálja!) Hogyan biztosítható az adatkonzisztencia? Mit jelent, hogy egy adatátvitel védtelen / nem determinisztikus? Hogyan hat egy adatátvitel determinisztikusságának biztosítása a látenciára (válaszát indokolja)?



- Determinisztikus (deterministic): Az átvitel időzítése determinisztikus. Ez több időbe, azaz látenciába (latency) kerül, így lassabb.
- Nem determinisztikus (nondeterministic): Az átvitel időzítése nem determinisztikus. Itt hamarabb érvényre jut a számított adat, így gyorsabb.

Az adatkonzisztencia a "Rate Tranzition" blokk alkalmazásával oldható meg. A determinisztikusság betartása a látenciát növeli, mivel a gyorsabb taszk által generált adat a lassabb taszknak csak a következő ütemére jut érvényre.

V. State Charts és State flow

1. Hasonlítsa össze a StateCharts ÉS és VAGY alállapotait! Milyen lehetőségek vannak az ÉS alállapotok működésének szinkronizálására?

- VAGY alállapotok: Egyszerre csak az egyik alállapot lehet aktív.
- ÉS alállapotok: Mindegyik aktív lehet.

Az ÉS állapotok szinkronizációja megoldható például egy közös fő állapotba való helyezéssel, vagy állapotra vonatkozó feltételek – in() – alkalmazásával.

2. Mi egy átmenet címkéjének szintaktikája Stateflow-ban? Magyarázza el az egyes elemek jelentését!

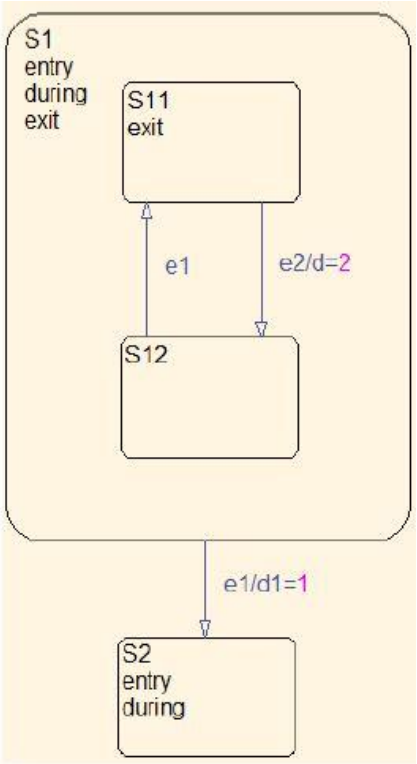
Az átmenet szintaktikája: *ESEMÉNY [Feltétel] {Feltételes művelet} / MŰVELET*

- ESEMÉNY: Az átmenet az adott esemény hatására megy végbe (lehet ε is).
- Feltétel: Az átmenet csak akkor megy végbe, ha a feltétel igaz.
- Feltételes művelet: Mindig végrehajtódik, ha a Feltétel igaznak értékelődik ki.
- MŰVELET: Műveletek halmaza, melyek csak akkor hajtódnak végre, ha az átmenet engedélyezett.

3. Adja meg, hogy mi történik és milyen lépésekben az alábbi Stateflow-diagram végrehajtásakor az e1 esemény hatására!

Először is tudni kellene, hogy éppen melyik állapot aktív.
Tegyük fel, hogy az S1 főállapot és az S11 alállapot volt aktív.

1. Kilépés az S1 állapotból
    1. Kilépés az S11 állapotból
        1. S11 exit műveletének végrehajtása
        2. S11 inaktív
    2. S1 exit műveletének végrehajtása
    3. S1 inaktív
2. Az átmenet műveletének végrehajtása (d1 = 1)
3. Belépés az S2 állapotba
    1. S2 aktív
    2. S2 entry műveletének végrehajtása



VI. LabVIEW

1. Mi a különbség a LabVIEW tömb (array) és cluster adatszerkezete között? Mi az auto-indexelt shift-regiszter (auto-indexed shift register) konstrukció? Hozzon egy példát, amikor ennek használata előnyös lehet!

A tömb homogén adatszerkezet, míg a cluster struktúra jellegű, így inhomogén is lehet.

Auto-indexelt shift-regiszter:

- Műveletvégzés egy tömb elemein ciklus segítségével.
- A bemeneti alagúton a külvilág felé tömb, a ciklusmag felé a ciklusszámláló által kiválasztott elem (olvasás).
- A kimeneti alagúton a ciklusmag felé a ciklusszámláló által kiválasztott elem (írás), a külvilág felé a tömb, de csak miután leállt a ciklus!
- Akkor lehet ennek a használata előnyös, ha a későbbiekben fel akarjuk még használni minden iteráció eredményét más műveletekhez.

2. Hasonlítsa össze a LabVIEW-ban hurkok közötti adatcserére használt változók típusait! Ismertesse részletesen a Functional Global Variable felépítését és működési elvét!

Hurkok közötti adatcseréhez használt változók típusai:

Változó	Hatókör	Kötés	Jellegzetesség
Lokális (Local)	Egy VI-on belül	Az előlap egy eleméhez	Elsősorban hurkok közötti adatmozgatásra
Megosztott (Shared)	Egy targeten / Hálózaton	Targethez	Projekt használata szükséges
Globális (Global)	Egy targeten	Speciális VI elemeihez (csak előlap)	Nincs szükség projekt használatára
Functional Global Variable	Egy targeten	SubVI Shift regiszteréhez	Nem igazi változó, funkciókkal bővíthető

Functional Global Variable felépítése:

- Ez nem más mint egy subVI, aminek a bemenete és a kimenete egy darab dróttal van összekötve egy while cikluson keresztül (shift regiszterrel), aminek a leállási feltételére konstans true értéket kötöttünk.
- Amikor új adat jelenik meg a bemeneten, a ciklus egyszer lefut és eltárolja a változót. Mivel a subVI nem reentráns, ezért mindegyik példánya mögött ugyanaz a kód fut, így a változók értékei megmaradnak.

3. Hasonlítsa össze a LabVIEW-ban rendelkezésre álló szinkronizációs módszereket! Ismertesse részletesen a szinkronizálás menetét Notifier használatával!

Szinkronizációs módszerek összehasonlítása:

Módszer	Átadható adat	Versenyhelyzet	Overhead	Buffer
Változó	Tetszőleges	Lehetséges	Jelentős	Nincs
Notifier	Tetszőleges	Kizárt	Nincs	Nincs
FIFO (Queue)	Tetszőleges	Kizárt	Nincs	Van

A notifier objektumot először létre kell hozni, majd ha mindenki befejezte a használatát, akkor törölni kell. A működése igen egyszerű. Legyen két folyamat A és B. Az A folyamathoz szinkronizáljuk hozzá B folyamatot. Amikor az A folyamat úgy ítéli meg, hogy most kell B nek lefutnia, küld egy notifier-t, hogy "most futhat". A B folyamat végrehajtását pedig egy ilyen notifier érkezéséhez kötjük.

4. Mire szolgál a LabVIEW Event Structure objektuma és hogyan működik? Mi a különbség a Notify és Filter események között?

Az Event Structure objektum események lekezelésére szolgál. A rendszer események dinamikusan tárolódnak. Az objektumnak előre meg kell mondani milyen események figyelésére "iratkozzon fel". Amikor az esemény bekövetkezik, akkor lekezeljük, majd töröljük.

Notify esemény során a LabVIEW csak értesít arról, hogy volt ilyen esemény, de már lekezelte, Filter esemény során még a LabVIEW előtt lekezelhetjük az eseményt, és ha töröljük, akkor a LabVIEW nem is fog foglalkozni vele.

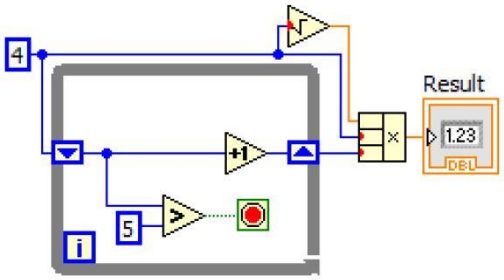
5. Ismertesse a Property Node és Invoke Node fogalmát és hozzon példát használatukra! Mire szolgál a VI server?

A property node segítségével lekérdezhetjük, illetve beállíthatjuk az egyes objektumok paramétereit. Egyszerre akár többet is.
Az Invoke Node segítségével a bekötött objektum metódusait hívhatjuk meg, egyszerre csak egyet.

A VI-server lehetővé teszi, hogy programozottan, platformfüggetlenül OO módom hozzáférjünk a LabVIEW-hoz, vagy a futtatási környezet egyéb objektumaihoz (Active X).

6. Mi lesz látható a Result indikátoron az alábbi blokkdiagram futtatása után? Válaszát indokolja!

A szorzóblokk 3 számot szoroz össze, ebből a gyök 4 és a 4 még a ciklus lefutása előtt rendelkezésre áll. A ciklus a bemenetére kapott 4-es kezdeti értékű változót addig inkrementálja, amíg a shift regiszter előző értéke 6 nem lesz. Mivel a leállási feltétel, csak a ciklus legvégén értékelődik ki, így az inkrementálás még egyszer bekövetkezik, tehát a kimenet 7 lesz, majd leáll a ciklus.
A végeredmény ezáltal 2*4*7 = 56.



VII. LabVIEW Control Design

1. Milyen reprezentációs módok érhetők el LabVIEW-ban LTI rendszerek leírására? Milyen paraméterei vannak az átviteli függvénnyel történő leírásnak? Mi a különbség a numerikus (Numeric) és paraméteres (Symbolic) megadási módok között?

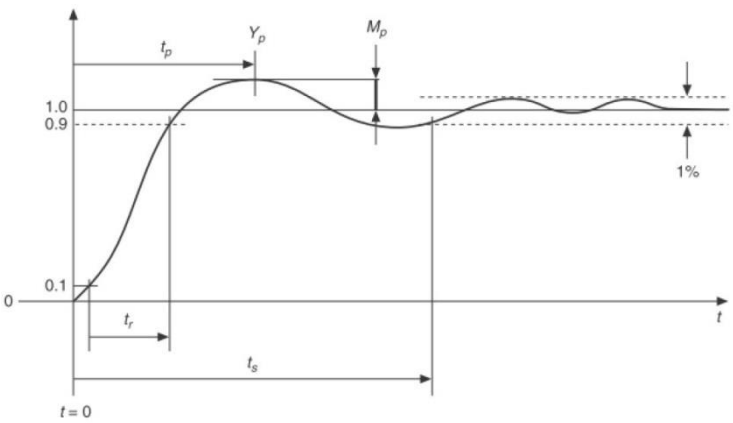
Reprezentáció	Folytonos	Diszkrét
Átviteli függvény (Transfer Function)	$W(s) = \frac{b_0 + b_1 s + \ \dots + b_m s^m}{a_0 + a_1 s + \ \dots + a_n s^n}$	$D(z) = \frac{b_0 + b_1 z + \ \dots + b_m z^m}{a_0 + a_1 z + \ \dots + a_n z^n}$
Zérus-pólus-erősítés (Zero-Pole-Gain)	$W(s) = k\frac{(s-z_1)(s-z_2)\dots(s-z_m)}{(s-p_1)(s-p_2)\dots(s-p_n)}$	$D(z) = k\frac{(z-z_1)(z-z_2)\dots(z-z_m)}{(z-p_1)(z-p_2)\dots(z-p_n)}$
Állapotteres alak (State- Space)	$\dot{x} = Ax + Bu$ $y = Cx + Du$	$x_{k+1} = Ax_k + Bu_k$ $y_k = Cx_k + Du_k$

TF paraméterei:
- Számláló polinom (1D array - SISO)
- Nevező polinom (1D array - SISO)
- Mintavételi idő (numeric)
- Holtidő (numeric)

Szimbolikus megadásnál egy clusterben meg kell adni pluszban a paraméter név – érték párosokat. Ezen paraméter nevekre hivatkozva szimbolikusan, sztring tömbökkel kell megadni a számláló és nevező vektorokat – műveleteket is bele lehet írni (b*k-2*m).
A paraméterek értéke controllhoz is köthető!

2. Mire használható a CD Parametric Time Response blokk? Milyen lehetőségeket nyújt a LabVIEW Control Design toolkit a frekvenciatartománybeli analízisre?

A CD Parametric Time Response blokk egy rendszer minőségi jellemzőit számítja ki, egy beállított gerjesztés hatására (ugrászválasz, impulzusválasz, szabad válasz):
- Rise Time, t_r
- Peak Time, t_p
- Settling Time, t_s
- Overshoot (%), M_p
- Stady-State Gain
- Peak value, y_p



Frekvenciatartománybeli analízis lehetőségei:
- Bode-diagramm
- Nyquist-diagram
- Pólus-zérus elrendezés
- Gyökhelygörbe

3. Ismertesse a LabVIEW-ban implementált analitikus diszkrét PID-tervezési algoritmus elvét!

Adott:
- A szakasz
- Követelmények: Minimális fázis- és erősítéstartaltalék

Keresett:
- Ki, Kp, Kd paramétereket, melyek mellett a zárt kör stabil és teljesülnek a követelmények.

$$W_{PID}(s) = K_p + \frac{K_i}{s} + K_d s$$

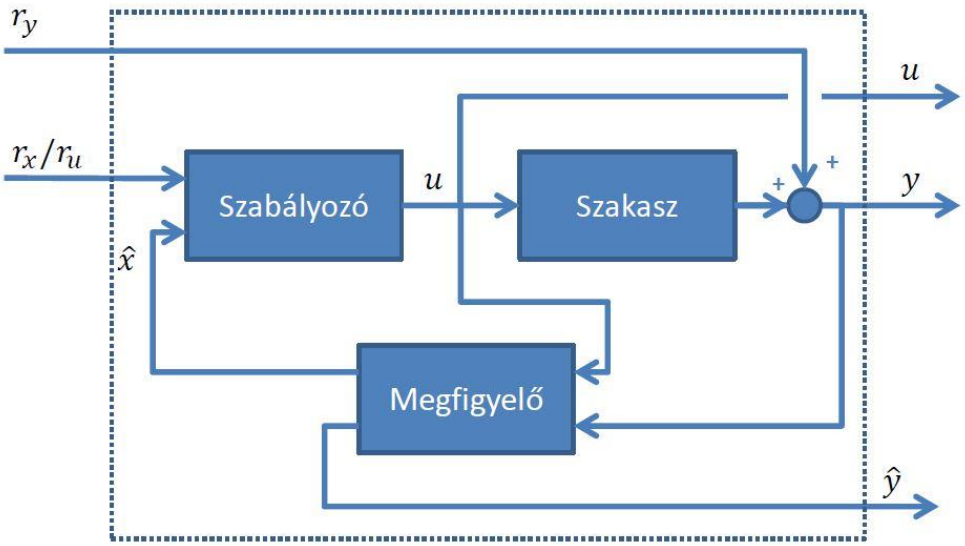
$$D_{PID}(z) = \frac{K_2 \cdot z^2 + K_1 \cdot z + K_0}{z^2 - 1}$$

$$K_3 := K_2 - K_0$$

Analitikus PID-tervezési algoritmus lépései:
1. Keressük a nem instabil kört lehetővé tevő K_3 tartományt.
2. Osszuk fel ezt a K_3 tartományt adott számú pontra.
3. Minden K_3 értékhez határozzuk meg, hogy ott milyen $K_1 – K_2$ tartományok kielégítőek.
4. Minden tartományon belül válasszunk N pontot véletlenszerűen, egyenletes eloszlással.
5. Az összes pont (K_1, K_2, K_3) koordinátáját számoljuk át (K_p, K_i, K_d) koordinátákra.
6. Válasszunk egyetlenmegoldást az összes pont centroidjaként.

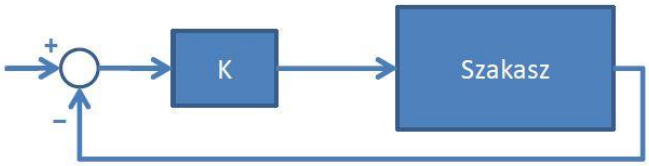
4. Mire szolgál a LabVIEW CD Ackermann blokkja? Adja meg a System Included With Noise beállítás mellett az állapotteres szabályozási kör blokkvázlatát, a LabVIEW-ban létrehozott kör be- és kimeneteit!

A CD Ackermann blokk az állapotteres tervezésnél használt Ackerman képlet segítségével (illetve MIMO esetben annak általánosításával) számított erősítésvektort és zárt kör pólusokat adja meg.



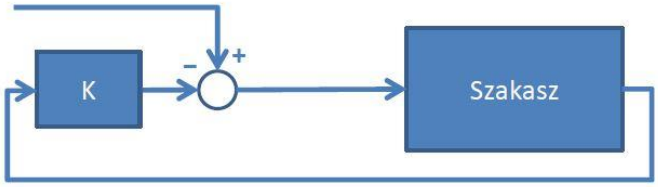
5. Mi a különbség a LabVIEW Control Design Toolkit állapotteres szabályozási köreinek Regulator és Compensator típusai között? Melyik esetében valósítható meg az alapjel miatti korrekció és hogyan?

Compensator: alapjel az állapotra vonatkozik (r_x)



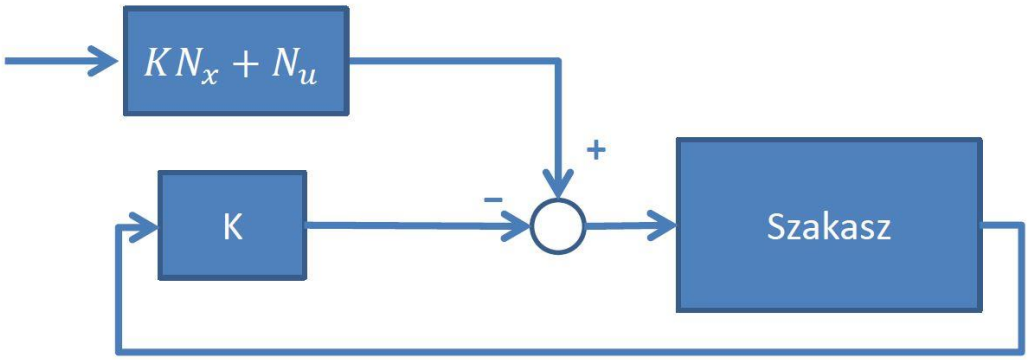
$$u = K(r_x - x)$$
$$\dot{x} = (A - BK)x + BKr_x$$

Regulator: alapjel a beavatkozó jelre vonatkozik (r_u)



$$u = r_u - Kx$$
$$\dot{x} = (A - BK)x + Br_u$$

Csak Regulator esetben valósítható meg alapjel miatti korrekció:



6. Mire szolgál a LabVIEW Simulation Loop blokkja és milyen beállítási lehetőségeket kínál? Milyen elemek helyezhetők el ebben a hurokban és milyen időzítéssel?

A Simulation Loop a Timed Loop egy speciális változata, melynél a már ismert hurok tulajdonságok érvényesek.

Beállítható:
- Szimuláció kezdeti ideje
- Szimuláció végideje
- Solver (algoritmus, lépésközök, tolerancia)
- Diszkrét lépésköz

A hurokba helyezhetőek:
- Szimulációs blokkok (mindenképpen)
- Emellett szinte bármi, akár subVI is
- NEM lehet: for/while/case…

7. Milyen lehetőségeket kínál a LabVIEW egy szabályozó implementálására? Mik ezeknek a módszereknek az előnyei és hátrányai?

Control and Simulation Loop:

- Simulation Loop -ban tetszőleges szimulációs modell elhelyezhető
- Lehetőség folytonos idejű rendszerek implementálására
- Csak állandó lépésközű solver használható
- A szimulációnak overhead-je van

MathScript RT:

- Képes valósidejű működésre
- A felhasználható függvények köre korlátozott
- A kódot minden ciklusban értelmezni kell és végre kell hajtani

„saját” megvalósítás:

- A szabályozó megvalósítása alapelemekből:
    - Szorzás (mátrix)
    - Összeadás
    - Késleltetés
- Kényelmetlen, de teljesen kézben tartható
- Nincs számottevő overhead

VIII. LabVIEW Real-Time

1. Adja meg a valósidejűség, a determinizmus és a jitter értelmezését! Mi okozhat jittert valósidejű LabVIEW-alkalmazásokban? Hozzon példát jitter-t okozó műveletekre!

Valósidejű rendszer: Egy eseményre mindig, garantáltan reagál egy adott időn belül.
Determinizmus: Mennyire konzisztens a rendszer válaszának időbelisége.
Jitter: A ciklusidőtől való eltérés.

Jittert okozhat például egy ciklus futásidejének ingadozásából adódó ciklusidő ingadozás. Ezt okozhatja például, ha engedélyezzük a megszakítások kiszolgálását, vagy közös erőforrások használata.

2. Milyen elven működik a LabVIEW RT ütemező? Hogyan valósítható meg egyetlen VI-on belül egy determinisztikus hurokra és több nem determinisztikus hurokra osztott alkalmazás? Hogyan lehet a nem determinisztikus hurkok periodikus végrehajtását biztosítani?

A LabVIEW RT Ütemező kétféle működést biztosít: preemptív és round robin. Az egy VI-on belül több hurok futtatása megoldható Multithreading megvalósítással.
Nem determinisztikus hurkok periodikus végrehajtását biztosíthatjuk időzített hurokkal, vagy blokkoló függvény használatával (pl timeout).

3. Állítsa prioritás szerint csökkenő sorrendbe az alábbi szálakat:

- Magas (high) prioritású VI-ban megvalósított While hurok 5
- Idő-kritikus (time-critical) prioritású VI 2
- NI Scan engine 1
- Normál (normal) prioritású VI-ban megvalósított időzített hurok 4
- RT OS ütemező 3

(Nagyobb szám, alacsonyabb prioritás)

4. Milyen lehetőségek vannak a valósidejű alkalmazás determinisztikus és nem determinisztikus hurkai közötti kommunikációra? Hasonlítsa össze ezeket a lehetőségeket!

Elosztott változó (singleprocess) RTFIFO-val:
- RT működésre optimalizált
- Single Process Shared Variable
- Csak fix hosszúságú adattípusok (tömb még lehet)
- Időbélyeggel ellátható

Többelemű elosztott változó RTFIFO-val:
- "valódi FIFO"
- Megadható elemszám, ne essünk túlzásokba!
- Alul/felül csordulhat

5. Milyen lehetőségek vannak a valósidejű alkalmazás és a host alkalmazás közötti kommunikációra? Hasonlítsa össze ezeket a lehetőségeket!

Front Panel kommunikáció:

- Az RTtargeten futó VI-hoz is rendelhető frontpanel
- A front panel a host-on megjeleníthető
- Nagy overhead, csak fejlesztési és debug célra ajánlott

Network-published elosztott változók

- Kommunikáció a determinisztikus hurokkal: RTFIFO használata szükséges
- Kommunikáció nem-determinisztikus hurkokkal: RT FIFO használata nem szükséges
- Használata egyszerű és kényelmes

Network stream

- Veszteségmentes kommunikáció
- Egyirányú
- Egy író–egy olvasó
- Bufferelt
- Nem determinisztikus

Standard protokollok

- Adatok továbbítása nem LabVIEW-alapú eszközök felé
- TCP
- UDP