5. Digitális rendszertervezés IV. - Esettanulmány - Ring-oszcillátor-alapú hőmérséklet-érzékelő

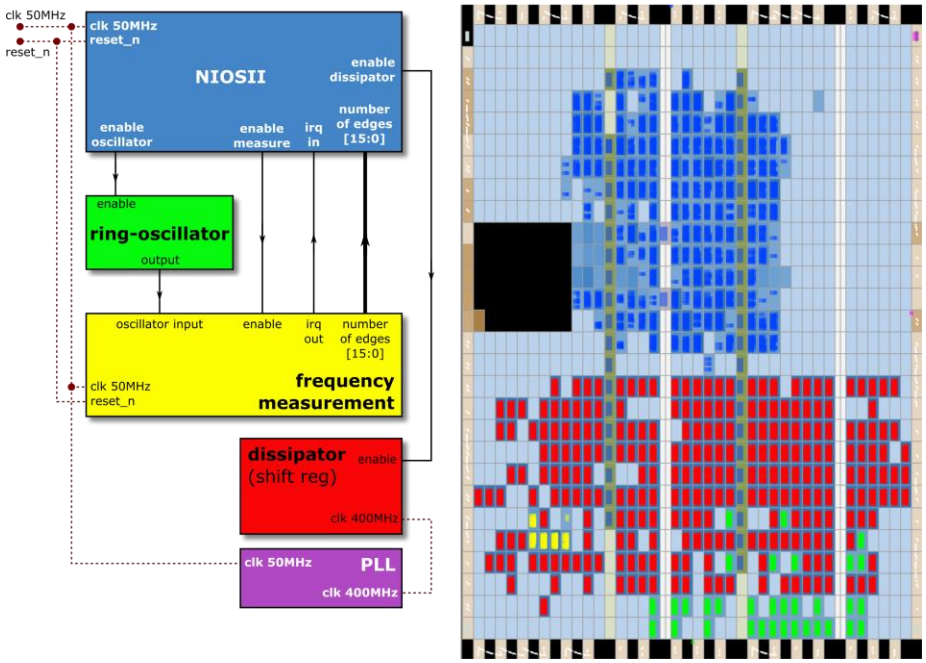
Szerző: Horváth Péter

A *Ring-oszcillátor-alapú hőmérséklet-érzékelő* c. laboratóriumi gyakorlat során egy előre elkészített rendszert vizsgálunk meg, amelynek összetevői a korábbi laboratóriumok során megismert módszerekkel készültek és amelynek elsődleges célja, hogy egy kézzelfogható példán keresztül szemléltesse a CMOS áramkörök hőmérsékletfüggő viselkedését.

Az FPGA-t felépítő CMOS áramkörök hőmérsékletfüggését arra fogjuk felhasználni, hogy egy hőmérséklet-érzékelő szenzort hozzunk létre. Mivel maga a hőmérsékletfüggő elem és a hozzá tartozó kiolvasó és adatfeldolgozó rendszer egyaránt egy FPGA-n belül valósul meg, egyszerűsége ellenére a példarendszer SoC áramkörnek tekinthető.

5.1. A rendszer hardver komponensei

A hardverplatform öt komponensből áll, amelyek közül egy a rendszer órajeleit előállító PLL, három a hőmérséklet-érzékelést és kiolvasást valósítja meg, egy pedig - egyfajta on-chip tesztkörnyezetként - a chip hőmérsékletének nagymértékű befolyásolására képes disszipáló elem, amely a mérés során ki/be kapcsolható. A 5-1. ábra a rendszer felépítését szemlélteti. A 5-1. ábra jobb oldalán az egyes komponensek fizikai elhelyezkedése figyelhető meg a *Quartus II* fejlesztőkörnyezet *Chip Planner* felületén. A rendszer összetevői színük alapján azonosíthatók.



5-1. ábra A példarendszer felépítése

A hőmérsékletfüggő elem egy **ring-oszcillátor**, amelyről részletesen a következő pontban lehet olvasni. A ring-oszcillátor által előállított négyszögjelnek *(oscillator_input)* a **hőmérséklettől függő frekvenciáját** egy egyedi funkcionális egység, a **frekvenciamérő elem** *(frequency measurement)* alakítja a Nios II rendszer által feldolgozható formára a következőképpen: A frekvenciamérő elem **egy 10 ms-os időablakban megszámolja a ring-oszcillátor által generált négyszögjel felfutó éleit**, majd a számláló értékét a Nios II rendszer megfelelő bementi portjára *(number of edges)* kapcsolja. Ezután megszakítást generál a Nios II processzor felé, amely ennek hatására a számláló értékét beolvassa, majd a JTAG UART modulon keresztül a host számítógép terminálablakában jeleníti meg azt. A megjelenített érték természetesen további feldolgozást és karakterizációt igényel, ha az abszolút hőmérsékletértékekre vagyunk kíváncsiak. Ezt az utófeldolgozást a példarendszer nem implementálja, mivel a hőmérsékletfüggő viselkedés kvalitatív szemléltetéséhez ez már nem elengedhetetlen.

5.1.1. Hőmérsékletfüggő elem

A rendszer alapja egy ring-oszcillátor, amelynek kimeneti frekvenciája a hőmérséklettel változik. A ring-oszcillátor egy visszacsatolt inverterlánc, amely megvalósítható az FPGA általános célú erőforrásainak részét képező LUT-ok felhasználásával. A 5-2. ábra a ring-oszcillátor VHDL modelljét mutatja.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity ring_oscillator is
5      generic (oscillator_size: integer);                    -- Az oszcillátor invertereinek a száma szintézis paraméter.
6      port (enable_osc:         in  std_logic;
7            osc_output:         out std_logic);
8  end entity ring_oscillator;
9
10 architecture rtl of ring_oscillator is                     -- Az egyes inverterek közötti vezetékek egyetlen vektor elemeit alkotják.
11
12     signal osc_wires:   std_logic_vector (oscillator_size downto 0);
13     attribute syn_keep: boolean;                           -- Expliciten utasítanunk kell a szintézert, hogy ne optimalizálja ki az inverterláncot.
14     attribute syn_keep of osc_wires: signal is true;
15
16 begin
17
18     L_GEN_OSC: for i in 1 to oscillator_size generate      -- A sok egyforma áramköri elemet a VHDL generate utasításával példányosítjuk.
19         osc_wires(i) <= not osc_wires(i-1);
20     end generate;
21     osc_wires(0) <= not (osc_wires(oscillator_size) and enable_osc);   -- Az inverterlánc egyik eleme valójában egy NAND kapu, amely engedélyező funkciót lát el.
22     osc_output <= osc_wires(oscillator_size);
23
24 end architecture rtl;
```

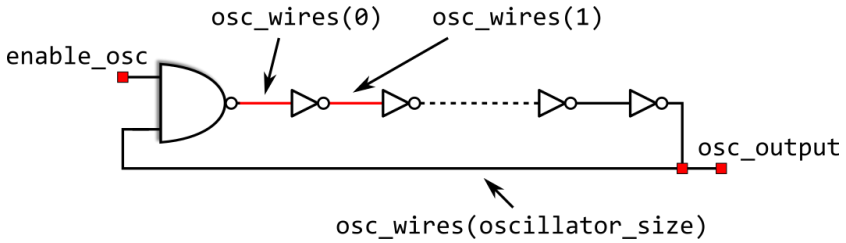
5-2. ábra A ring-oszcillátor VHDL modellje

A ring-oszcillátor frekvenciája nemcsak a hőmérséklettől, hanem a benne található inverterek számától is függ. A 5-2. ábrán bemutatott implementációban az inverterlánc hossza generikus

paraméter. Az általunk használt FPGA-ban az egyes invertereket megvalósító LUT-ok késleltetése annyira kicsi, hogy a MHz nagyságrendű kimenti frekvencia eléréséhez többszáz példányt kell láncba kapcsolnunk. Ennek a rengeteg áramköri elemnek a VHDL modellben való példányosítása körülményes lenne - és mert a generikus inverterszám miatt nem is ismerjük az elhelyezendő inverterek pontos számát -, ezért a példányosítást a VHDL **generate** utasításával végezzük, amely tulajdonképpen egy statikusan kiértékelt ciklus, amelynek törzsében parametrikusan megfogalmazott példányosítás utasítások vannak.

Az *osc_wires* azonosítóval az inverterek közötti vezetékek halmazára hivatkozunk. Mivel egy aszinkron módon visszacsatolt hálózat (kombinációs hurok) az alapvetően digitális funkció megvalósítására kifejleszett FPGA környezetben funkcionálisan értelmetlen (valójában kritikus hibának számít a kombinációs hurok létrehozása, ezt látni fogjuk a szintézer által generált log fájlban is), a szintézis eszköz az áramkört teljes egészében kioptimalizálná. Annak érdekében, hogy a szintézis algoritmust rákényszerítsük az általunk leírt áramkör megvalósítására, ún. **attribútumokat** rendelünk az érintett vezetékekhez. Ezeket az attribútumokat (vagy más néven szintézis pragmákat) - jelentésüket és lehetséges értékeiket - a szintézis szoftver határozza meg.

Az inverterek közül az egyik valójában egy NAND kapu, amely tulajdonképpen egy engedélyezhető inverter. Ha A bemenetére logikai alacsony szintet kapcsolunk, akkor a kimenete a B bemenet értékétől függetlenül logikai magas szint. Ha azonban az A bemenetre logikai magas szintet kapcsolunk, akkor a kimenet a B bementre vonatkoztatva inverterként viselkedik. A NAND kaput az inverterláncba beiktatva a NAND kapu A bemenetének változtatásával válthatunk stabil és instabil, vagyis kikapcsolt és oszcilláló állapot között. A 5-3. ábra a 5-2. ábra által leírt ring-oszcillátor logikai kapcsolási sémáját mutatja.

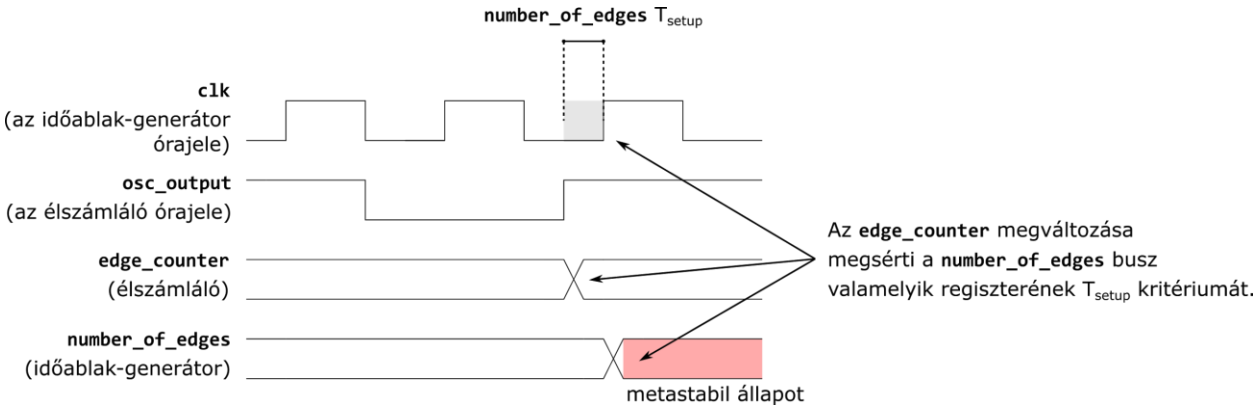


5-3. ábra A ring-oszcillátor logikai kapcsolási sémája

5.1.2. Frekvenciamérő elem

A ring-oszcillátor kimeneti jelének éleit egy 10 ms-os időablakban megszámláló frekvenciamérő elem három almodulból áll. A 10 ms-os időablakot és az élszámlálást egy-egy számláló valósítja meg, amelyeket egy órajelszinkronizáló egység kapcsol össze egymással. Ez utóbbira azért van szükség, mivel az élszámlálást az időablak-generátor engedélyezi/tiltja és az élszámláló értékét az időablak-generátor az időablak végén kiolvassa. A két számláló között tehát adatcsere történik. A probléma az, hogy a két számláló áramkör egymástól független, ún. **aszinkron órajeltartományokban** működik (*Clock Domain Crossing*, CDC). Az élszámlálót a

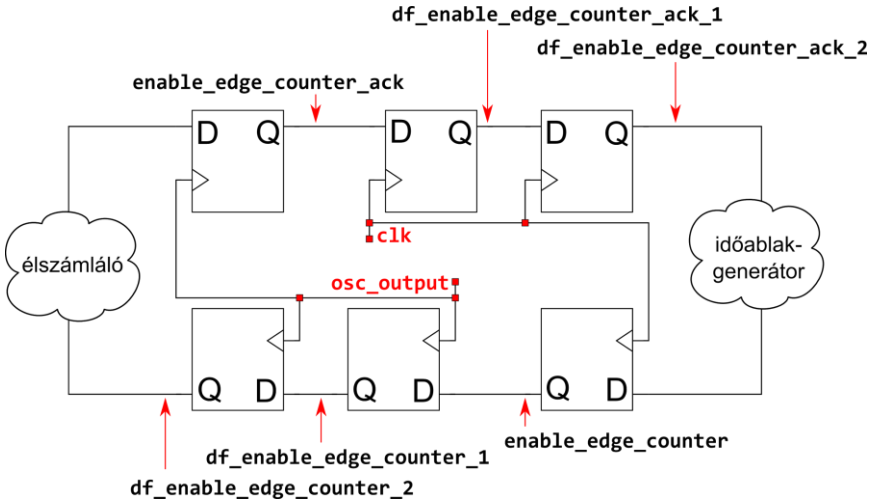
mérendő jel hajtja, míg az időablak-generátor egy ismert frekvenciájú órajelről jár. A két órajel közötti **frekvencia- és fázisviszonyok előre nem ismertek**, ráadásul a mérendő jel változása miatt folyamatosan változnak is. A problémát a 5-4. ábra szemlélteti.



5-4. ábra Metastabil állapot kialakulása az aszinkron órajeltartományok határán

Mivel a két órajeltartomány *(clk, osc_output)* frekvencia- és fázisviszonya folyamatosan változik, nem garantálható az, hogy amikor az időablak-generátor mintavételezi az élszámlálót, akkor ez utóbbi nem változik meg, megsértve ezzel az időablak-generátorban az élszámláló értékét eltároló regiszterek valamelyikének *(number_of_edges)* ***Tsetup*** kritériumát. Ha megsérti, akkor a *number_of_edges* érintett **regiszterei metastabil állapotba kerülhetnek**, ami azt jelenti, hogy kimenetük feszültségszintje nem tekinthető **sem logikai alacsony**, **sem magas** szintnek. Az e regisztereket olvasó egyéb áramköri elemek **egy része nullaként**, **más része egyként értelmezheti** a metastabil állapotban lévő regiszter kimenetét, ami értelemszerűen hibás működéshez vezethet. A metastabil állapot - ahogy neve is mutatja - nem stabil, tehát a **környezeti zaj hatására hamar stabilizálódik**, és az érintett regiszter beáll valamelyik logikai szintre.

A probléma megoldására a bemutatott rendszer ***hand-shake*** kapcsolatot valósít meg az élszámláló és az időablak-generátor között. A hand-shake kommunikáció lehetővé teszi, hogy az élszámláló kimeneti regiszetereinek értéke semmiképpen se változzon meg, mialatt az időablak-generátor mintavételezi őket. Ezzel azonban a probléma nem szűnt meg teljesen, hiszen most a hand-shake jelekre vonatkozóan lehet ugyanazt elmondani, amit korábban a mintavételezett buszról állítottunk, hiszen mindegyik jel két aszinkron órajeltartomány határán visz át információt. A probléma az ún. ***double-flopping*** technikával kezelhető, amelyet a 5-5. ábra szemléltet.



5-5. ábra Double-flopping

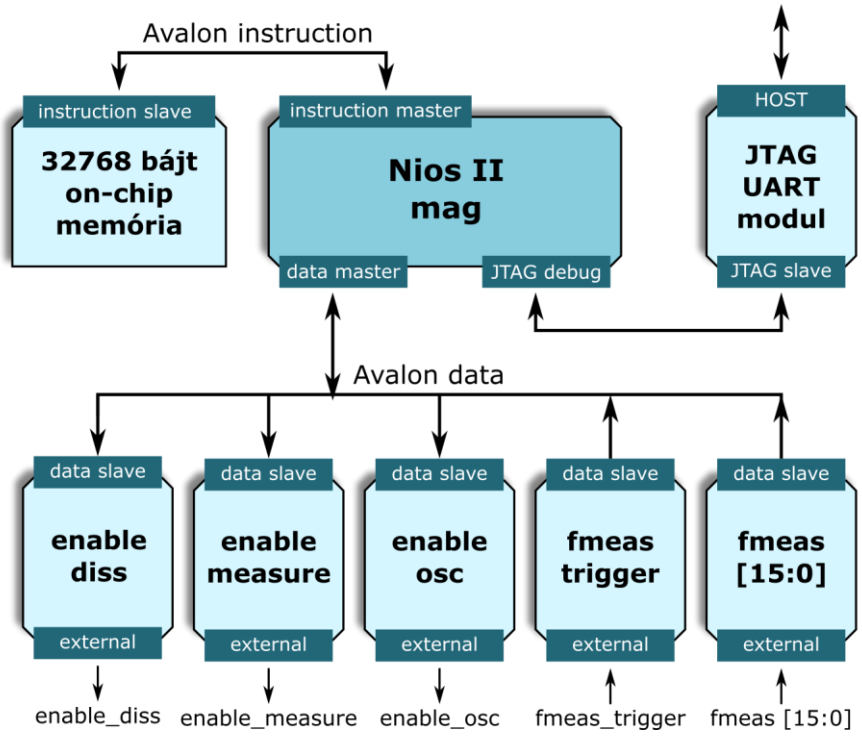
A 5-5. ábrán látható megoldás lényege, hogy az egyik órajeltartományból a másikba adatot átvivő jeleket **két szinkronizáló flip-flopon vezetjük keresztül**. Figyeljük meg, hogy **a szinkronizáló flip-flopok mindig a fogadó oldal órajelét kapják**. Valójában ez a megoldás nem akadályozza meg a metastabil állapot kialakulását, hiszen – a 5-4. ábra felső ágának példáját vizsgálva - az *enable_edge_counter_ack* jelen előállhat metastabil állapot, amit azonban CSAK EGY áramköri elem, az első szinkronizáló flip-flop olvas be, így nem áll fenn az a veszély, hogy más-más áramköri elemek különbözőképpen értelmezik a metastabil állapotra jellemző feszültségszintet. Az első szinkronizáló flip-flop vagy logikai nullaként, vagy egyként értelmezi azt. Természetesen előfordulhat, hogy a komparálási szinthez közeli érték miatt a szinkronizáló flip-flop maga is metastabil állapotba kerül, de ennek nagyon kicsi a valószínűsége, mivel az *enable_edge_counter_ack* vonalon kialakult metastabil állapotnak egy egész órajelciklus áll rendelkezésére, hogy stabilizálódjon, ami általában elég. Ha mégis előfordulna, hogy az első szinkronizáló flip-flop is metastabil állapotba kerül, akkor a második fokozat további egy órajelciklust biztosít a stabilizálódáshoz, még tovább csökkentve annak esélyét, hogy a metastabil állapot kijusson a szinkronizáló egység kimenetére. Nagy megbízhatóságú rendszerekben előfordul, hogy a két szinkronizáló flip-flop helyett négyet használnak. Ekkor a metastabilitásból eredő hibák valószínűsége gyakorlatilag nulla.

Felmerülhet a kérdés, hogy miért volt szükség egyáltalán a hand-shake kommunikáció megvalósítására, miért nem alkalmaztuk a double-flopping technikát közvetlenül az átvitt adatbuszon? Figyeljük meg, hogy **a double-flopping technika** miatt az átviendő adat - attól függően, hogy az első szinkronizáló flip-flop az esetlegesen előforduló metastabil állapotra jellemző feszültségszintet logikai magas vagy alacsony szintként értelmezi - **egy órajelciklus késleltetést iktathat be a kommunikációba**. A többlet órajelciklus megléte **nem prediktálható**! Ha az átviendő adatbusz minden bitjére kiépítjük a double-flopping rendszert, akkor nem biztosítható, hogy a busz egyes bitjei a fogadó oldalon is összetartozó adatokat képviseljenek.

A frekvenciamérő elem VHDL modellje a *tempsensor.vhd* forrásállományban található.

5.1.3. Adatfeldolgozó elem

A frekvenciamérő elem kimenete egy 16-bites adatbusz, amely mindig az oszcillátor kimeneti jelének a legutóbbi időablakban előforduló felfutó éleinek a számát jelenti. Ennek az értéknek az ismeretében az oszcillátor frekvenciája kiszámítható. A 16-bites adatbuszt egy Nios II processzoros rendszer olvassa be az ugyancsak a frekvenciamérő elem által az időablak végén generált megszakítás hatására. A Nios II rendszer felépítsét a 5-6. ábra mutatja be.



5-6. ábra A Nios II processzoros rendszer komponensei

A *Nios II beágyazott processzoros rendszerek* c. laboratóriumi gyakorlat során készített rendszerhez hasonlóan ez a példarendszer is tartalmaz egy JTAG UART modult a host számítógéppel való kapcsolattartás céljából, valamint egy 32 KB-os on-chip memóriát az adatok és utasítások számára. A rendszer I/O priféria a Nios II rendszer általános célú I/O vezérlőjével (PIO) vannak megvalósítva. Feladatuk a következő:

- ***enable_diss***: A disszipáló elem engedélyezése/tiltása.
- ***enable_measure***: A frekvenciamérő elem engedélyezése/tiltása.
- ***enable_osc***: Az oszcillátor engedélyezése/tiltása.
- ***fmeas_trigger***: Megszakításkérő bemenet, amelyet a frekvenciamérő elem hajt meg.
- ***fmeas [15:0]***: A frekvenciamérő elem 16-bites kimeneti busza.

5.1.4. Disszipáló elem

Annak érdekében, hogy szemléltetni tudjuk az áramköri elemek egymásra hatását egy chipen belül, a rendszer tartalmaz egy ki/be kapcsolható, visszacsatolt, 4096-bites shift-regisztert. A regiszterben tárolt érték az “101010…” mintát követi a kapcsolási aktivitás maximalizálása érdekében. Az elektromos és termikus hatás mértékének növelése érdekében a shift-regiszternek saját, nagyfrekvenciás órajelet állítunk elő (400 MHz). A shift-regiszter VHDL modellje a *dissipator.vhd* fájlban található.

5.1.5. Lábkiosztás

A példarendszer toplevel modulja az alábbi kievzetésekkel rendelkezik:

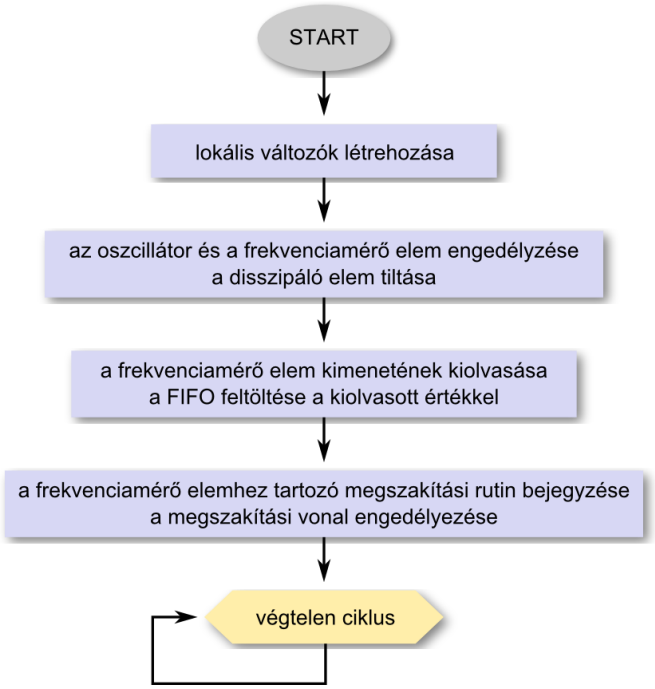
- ***clk***: Globális órajel bemenet, amelyet a DE0 fejlesztőkártyán lévő kristályoszcillátor hajt meg.
- ***reset_n***: Globális reset bemenet, amely a DE0 fejlesztőkártya BUTTON0 jelű nyomógombjára van kapcsolva.
- ***leds [7:0]***: 8-bites kimenet, amelynek egyes bitjei a DE0 fejlesztőkártya LEDG0-LEDG7 jelű LED-jeire vannak kapcsolva. A Nios II processzoron futó példaalkalmazás csak az alsó 3 bitet használja a következőképpen:
    - ***LEDG0***: Oszcillátor engedélyezve (1) / tiltva (0).
    - ***LEDG1***: Frekvenciamérő elem engedélyezve (1) / tiltva (0).
    - ***LEDG2***: Disszipáló elem engedélyezve (1) / tiltva (0).

5.2. A rendszer szoftver komponensei

A Nios II processzoron futó program egy főprogramból *(tempsensor.c: **`int main()`**)* és egy megszakítási rutinból *(isr.h: **`void isr_read_counter(void* context)`**)* áll.

5.2.1. A főprogram

A főprogram a rendszer erőforrásainak inicializálása és a frekvenciamérő elemhez tartozó megszakítási rutin bejegyzése után egy üres végtelen ciklusba lép. A végtelen ciklusba való belépés után a mikroprocesszor funkciója tulajdonképpen a megszakítási rutin időnkénti - a frekvenciamérő elem által küldött, kb. 10 ms-onkénti megszakításkérés hatására történő - lefuttatására korlátozódik. A főprogram folyamatábráját a 5-7. ábra mutatja.



5-7. ábra A főprogram folyamatábrája

A főprogram négy fordítási idejű paramétert tartalmaz, amelyeket preprocesszor-makrók valósítanak meg:

- ***AVERAGE_ON***: Ha értéke 1, akkor a frekvenciamérő elem által mért adatokon a processzoron futó program átlagolást végez, mielőtt az értékeket a kimenetre írná.
- ***WINDOW_SIZE***: Az átlagolás során használt ablakméretet határozza meg.
- ***SAMPLE_LIMIT***: A program számon tartja, hogy hány mintát olvasott ki a frekvenciamérő elemből. Ha a beolvasott minták száma eléri az e makró által definiált értéket, akkor a program letiltja a frekvenciamérő elem megszakítási vonalát, ami egyúttal a mérés végét is jelenti.
- ***DISSIPATOR_ON***: Ha értéke 1, akkor a mérés negyedénél *(SAMPLE_LIMIT/4)* a processzor aktiválja a disszipáló elemet, a mérés háromnegyedénél *(3*SAMPLE_LIMIT/4)* pedig deaktiválja azt. Így mind a bekapcsolás, mind a kikapcsolás hatása megfigyelhető a kimenetre írt mérési adatok megfelelő megjelenítésével.

A fenti funkciók mindegyikét a megszakítási rutin valósítja meg. A Nios II processzor esetén a megszakítási rutinok ***`void`*** visszatérési értékű függvények, amelyeknek paraméterlistája egyetlen, ***`void*`*** típusú változót tartalmaz. Ez a változó nem más, mint a megszakítási rutin kontextusára, vagyis a megszakítási rutin által igényelt információhalmazra mutató pointer.

Mivel maga a változó ***void**** típusú általános mutató, a mögöttes adatszerkezet bármi lehet. A frekvenciamérő elemhez tartozó megszakítási vonalhoz tartozó rutin esetén ez egy struktúra *(**isr_context**)*, amely minden olyan információt magában foglal, amelyre a megszakítás kiszolgálása során szükség van[1].

A megszakítási rutin bejegyzése az ***alt_irq_register()*** függvénnyel történik, amely a Nios II rendszerhez tartozó alacsony szintű driver-eket és magasabb szintű API funkciókat tartalmazó könyvtár, a HAL *(Hardware Abstraction Layer)* része. E függvénynek három paramétere van:

- ***alt_u32 id***: A Qsys alkalmazásban összeállított rendszer generálásakor minden megszakítási vonal egyedi azonosítót kap. Ez az azonosító egy makróként szerepel a mikroprocesszoros rendszer generálásakor előállított *system.h* fejállományban. A megszakítási rutin bejegyzésekor meg kell adnunk, hogy az éppen bejegyzett rutin melyik megszakítási vonalhoz tartozik.
- ***void* context***: Mivel a megszakítási rutint nem mi magunk fogjuk meghívni, hanem egy központi megszakításkezelő *(main ISR)*, amely mindegyik megszakítási vonal aktiválódásakor lefut, ezért előre meg kell adnunk, hogy az éppen bejegyzett, valamelyik konkrét megszakítási vonalhoz tartozó megszakításkezelő rutin hívásakor a központi megszakításkezelő milyen paramétereket (kontextust) adjon át az éppen bejegyzett megszakításkezelő rutinnak.
- ***alt_isr_func handler***: Maga a megszakításkezelő rutin egy függvénypointeren keresztül adódik át a bejegyzést végző függvénynek.

A frekvenciamérő elem által generált megszakításkérés kiszolgálásának előkészítése (a megszakítási rutin bejegyzése és a megfelelő megszakítás-engedélyező maszk beállítása) a főprogramban a 5-8. ábrán látható módon történik.

[1] Megjegyzendő, hogy a megszakítási rutin kontextusa globális változók segítségével is átadható lenne, de összetett, sok fordítási egységből álló rendszerek esetén ezek használata kerülendő.

```
-- isr.h --
8  typedef struct {
9      //...
17 } isr_context;

-- tempsensor.c: main() --
27 isr_context the_isr_context;
28 the_isr_context.value_base = (alt_u16*)FMEAS_BASE;
29 the_isr_context.fmeas_values = (alt_u16*)fmeas_values;
30 the_isr_context.window_size = WINDOW_SIZE;
31 the_isr_context.average_on = AVERAGE_ON;
32 the_isr_context.calls = (alt_u16*)(&isr_calls);
33 the_isr_context.sample_limit = SAMPLE_LIMIT;
34 the_isr_context.dissipator_on = DISSIPATOR_ON;
35 the_isr_context.enable_dissipator_base = (char*)(ENABLE_DISS_BASE);
36 alt_irq_register(FMEAS_TRIGGER_IRQ, (void*)&the_isr_context, isr_read_counter);
37 IOWR_ALTERA_AVALON_PIO_IRQ_MASK(FMEAS_TRIGGER_BASE, 0x1);
```

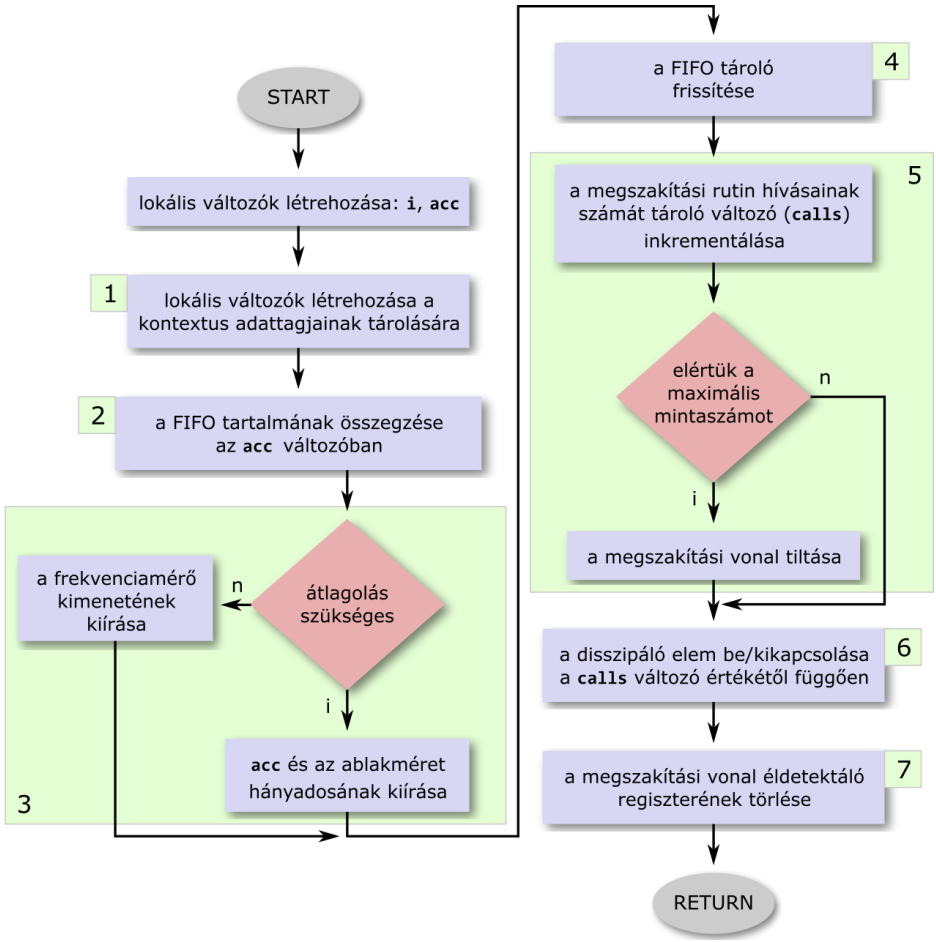
5-8. ábra A frekvenciamérő elem által küldött megszakításkérés lekezelésének előkészítése

5.2.2. A megszakítási rutin

A megszakítási rutin kontextusának elemei a következők:

- ***alt_u16* value_base***: A frekvenciamérő elem által szolgáltatott 16-bites érték címe. Az e memóriacímről beolvasott adat az oszcillátor kimenetének a legutóbbi időablakban érzékelt felfutó éleinek számát jelenti.
- ***alt_u16* fmeas_values***: A főprogram által létrehozott, *WINDOW_SIZE* méretű tömbre mutató pointer. Ez az a tömb, amelyet a megszakítási rutin FIFO tárolóként használ fel a frekvenciamérő elemtől kapott értékek ideiglenes megőrzésére.
- ***short window_size***: A *WINDOW_SIZE* makróban meghatározott érték.
- ***char average_on***: Az *AVERAGE_ON* makróban meghatározott érték.
- ***alt_u16* calls***: Ebben a változóban tárolódik, hogy a program futása során hányszor hívódott meg a megszakítási rutin. Gyakorlatilag az eddig beolvasott minták számát tárolja.
- ***short sample_limit***: A *SAMPLE_LIMIT* makróban meghatározott érték.
- ***char dissipator_on***: A *DISSIPATOR_ON* makróban meghatározott érték.
- ***char* enable_dissipator_base***: Az *ENABLE_DISSIPATOR* makróban meghatározott érték.

A megszakítási rutin egyszerűsített folyamatábrája a 5-9. ábrán látható.



5-9. ábra A megszakítási rutin folyamatábrája

A 5-10. ábrán a megszakítási rutinnak a 5-9. ábrán 1-gyel jelölt lépése látható.

```
25 alt_u16* value_base = ((isr_context*)(context))->value_base;
26 alt_u16* fmeas_values = ((isr_context*)(context))->fmeas_values;
27 short window_size = ((isr_context*)(context))->window_size;
28 char average_on = ((isr_context*)(context))->average_on;
29 alt_u16* calls = ((isr_context*)(context))->calls;
30 short sample_limit = ((isr_context*)(context))->sample_limit;
31 char dissipator_on = ((isr_context*)(context))->dissipator_on;
32 char* enable_dissipator_base = ((isr_context*)(context))->enable_dissipator_base;
```

5-10. ábra Lokális változók létrehozása a kontextus adattagjai számára

A paraméterlistán kapott struktúra (a megszakítási rutin kontextusa) minden elemét a magában a megszakítási rutinban létrehozott lokális változókba másoljuk. A lokális változók létrehozása nem elengedhetetlen. A kontextus adattagjait a megszakítási rutinban közvetlenül a paraméterlistából is kiolvashatnánk, a lokális változók létrehozása tehát csak a kód olvashatóságát javítja. Időzítéskritikus esetben az ilyen többlet műveletek beiktatása kerülendő.

A 5-11. ábrán a megszakítási rutinnak a 5-9. ábrán 2-vel jelölt lépése látható. A mozgóátlag kiszámításához a FIFO tároló teljes tartalmát összegezzük.

```
35 for (i = window_size-1; i >= 0 ; --i) {
36     acc += fmeas_values[i];
37 }
```

5-11. ábra A FIFO tartalmának összegzése az acc változóban

A 5-12. ábrán a megszakítási rutinnak a 5-9. ábrán 3-mal jelölt lépése látható. Az *AVERAGE_ON* makrótól függően (amelyet immár az ***average_on*** változón keresztül olvashatunk) vagy a FIFO-ban tárolt értékek átlagát, vagy magát a legfrissebb adatot írjuk ki a standard kimenetre. Jelen esetben a standard kimenet a JTAG UART modul. Az ide írt adatokat a fejlesztőkörnyezet a ***Console*** ablakban jeleníti meg. A HAL-beli ***IORD()*** függvény a paraméterlistáján megadott báziscím és az ugyancsak paraméterlistán megadott címeltolás alapján számított memóriacmről olvas be adatot. Mivel a PIO perifériatípushoz tartozó regiszterhalmaz első eleme maga az adat, ezért az eltolás ebben az esetben 0.

```
38 if ( average_on ) printf("%d; %d\n", *calls, (int)acc/window_size);
39 else printf("%d; %d\n", *calls, IORD(value_base, 0));
```

5-12. ábra Mérési adat kiírása a standard kimenetre

A 5-13. ábrán a megszakítási rutinnak a 5-9. ábrán 4-gyel jelölt lépése látható. A FIFO teljes tartalmát frissítjük (a benne tárolt értékeket eggyel shifteljük), majd a legújabb mérési adatot is eltároljuk benne.

```
42 for (i = window_size-1; i >= 1; --i) {
43     fmeas_values[i] = fmeas_values[i-1];
44 }
45 fmeas_values[0] = IORD(value_base, 0);
```

5-13. ábra A FIFO tartalmának frissítése

A 5-14. ábrán a megszakítási rutinnak a 5-9. ábrán 5-tel jelölt lépése látható. Az eddig beérkezett mérési adatok számát a ***calls*** változó tárolja. Ha a már eltárolt és feldolgozott mérési adatok száma elérte a *SAMPLE_LIMIT* makró értékét (amit a ***sample_limit*** változón keresztül érünk el), akkor a frekvenciamérőhöz tartozó megszakítási vonalat a HAL-beli ***alt_irq_disable()*** függvénnyel letiltjuk és a standard kimenetre egy a mérés befejezéséről szóló üzenetet helyezünk.

```
50 ++(*calls);
51 if ( sample_limit == *calls ) {
52     alt_irq_disable(FMEAS_TRIGGER_IRQ);
53     printf("FMEAS_TRIGGER disabled!\n");
54 }
```

5-14. ábra A megszakítási vonal tiltása a mérés végén

A 5-15. ábrán a megszakítási rutinnak a 5-9. ábrán 6-tal jelölt lépése látható. A megszakítási rutin a disszipáló elemet automatikusan kapcsolja be/ki a mérés során. A teljes mérés időtartamának negyedénél aktiválja, háromnegyedénél pedig deaktiválja azt.

```
57 if (dissipator_on && (*calls==(alt_u16)(sample_limit*0.25))) IOWR(enable_dissipator_base,0,0x1);
58 if (dissipator_on && (*calls==(alt_u16)(sample_limit*0.75))) IOWR(enable_dissipator_base,0,0x0);
```

5-15. ábra A disszipáló elem be/kikapcsolása

A 5-16. ábrán a megszakítási rutinnak a 5-9. ábrán 7-tel jelölt lépése látható. A frekvenciamérő elemhez tartozó megszakítási vonal élérzékeny, ami úgy valósul meg, hogy a megszakítási bemenetet a Nios II rendszeren belül egy éldetektáló áramkör fogadja. Az éldetektáló áramkör a megszakítási vonal felfutó élénél egy flip-flopot beállít, amelynek értékét a Nios II processzor már a belső, szintérzékeny megszaktási rendszerrel is érvényes megszakításként kezel. A megszakítási rutin végén ezt az éldetektáló flip-flopot törölni kell. A törlés a 5-16. ábrán látható függvényhívással lehetséges.

```
61 IOWR_ALTERA_AVALON_PIO_EDGE_CAP(FMEAS_TRIGGER_BASE, 1);
```

5-16. ábra Az éldetektáló flip-flop törlése