

Budapesti Műszaki- és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Rendes Martin

Perger Patrik Kristóf

**MMIO shift & add szorzó áramkör ACU soft-processzoros rendszerhez**

# Specifikáció

A cél egy tanszéki fejlesztési mikrokontrollerhez alkalmazás-specifikus perifériavezérlő fejlesztése. A félév során feladat az áramkör szintetizálható RTL modellje, funkcionális verifikációja és szintézise.

Az elkészítendő áramkör funkciója szorzás, shift & add algoritmus megvalósításával. A perifériának MMIO interfészen kell kommunikálnia. A két operandus mérete mindig azonos, maximális méretük 32 bit, ez legyen szintézis paraméter. Futásidőben lehessen konfigurálni, hogy a művelet végén legyen-e megszakítás generálás.

# Rendszerterv

A screenshot of a computer

AI-generated content may be incorrect.

## MMIO címek

A rendszert az alábbi memóriacímeken lehet elérni:

|  |  |  |
| --- | --- | --- |
| **Név** | **Mód** | **Leírás** |
| *operand* | W | A szorzótényezőket erre a címre kell beírni. Egyszerre 16 bit küldhető, így, ha az operandusok mérete nagyobb, mint 16 bit, kétszer kell bele írni, először az alsó 16 bitet, majd a további felső biteket. |
| *product\_1* | R | A szorzat alsó 16 bitje (0-15.) |
| *product\_2* | R | A szorzat 16-31. bitjei |
| *product\_3* | R | A szorzat 32-47. bitjei |
| *product\_4* | R | A szorzat felső 16 bitje (48-63.) |
| *ready* | R | Ha a megszakításgenerálás tiltva van, akkor ezen a címen olvasható, hogy vége van-e a műveletnek (0: folyamatban van; 1: vége, az eredmény kiolvasható) |
| *intr\_en* | W | A megszakításgenerálás[[1]](#footnote-1) ezen a címen tiltható (0) /engedélyezhető (1) |

## ACU MMIO peripheral adapter

Az áramkör a processzor MMIO interfésze (*address\_from\_acu, ready\_2\_acu, data\_from\_acu, stb.*) és a megvalósított logika (User Logic) közti kommunikációt valósítja meg. Az adapternek van egy szintézis-paramétere, az *operand\_size*. Ez a szám adja meg, hogy mekkorák legyenek az operandusok (maximum 32 bit). Ezt továbbítja az adatper a műveletvégzőnek az *operand\_size* kimeneten.

A processzor az operandusokat az *operand* címre küldi ki. Először az első operandus alsó 16 bitjét, majd (amennyiben van) a felső 16-ot. Ezt követően szintén erre a címre írja a második operandust. A modul egy belső állapotgéppel számlálja, hogy hányadik írás történt, így tudja megkülönböztetni, hogy melyik 16 bit érkezik. Amint megérkezik az utolsó 16 bites rész (ezt az *operand\_size* szintézis-paraméter alapján tudjuk felismerni), az adapter továbbítja ezeket a 32 bites *op\_1* és *op\_2* kimeneten, valamint egy *start* jelet a műveletvégzőnek. Az adapter ezt követően egy handshake-jelet (*start\_ack*) vár a logikától.

A megszakításgenerálás engelyézhető az *intr\_en* címre küldött 1-es bittel. Ebben az esetben az *interrupt\_enable* kimeneten egy 1-es érték jelenik meg.

Ha nincs engedélyezve a megszakításgenerálás, a processzor olvashatja a művelet állapotát a *ready* címen, ilyenkor az adapter a logika *ready* kimenetét jeleníti meg ezen a címen.

## User logic

Ez a modul végzi el a szorzást. Az alábbi be- és kimenetekkel rendelkezik:

|  |  |  |  |
| --- | --- | --- | --- |
| **Név** | **Irány** | **Méret** | **Leírás** |
| *start* | I | 1 | Az adapter ezen a porton jelzi, hogy a kimenetén olvashatók szorzótényezők, a művelet megkezdhető |
| *start\_ack* | O | 1 | Ezen a porton jelzi a logika, hogy megkezdte a művelet elvégzését |
| *intr\_enable* | I | 1 | Ha 1-es az értéke, akkor a művelet végén megszakítást generál a modul, ellenkező esetben a *ready* kimeneten jelzi a művelet állapotát. |
| *operand\_size* | I | 5 | Ezen a porton adja meg az adapter, hogy valójában hány bitesek az operandusok |
| *op\_1, op\_2* | I | 32 | A szorzótényezők |
| *product* | O | 64 | A szorzat |
| *Ready* | O | 1 | Ha tiltva van a megszakításgenerálás, ezen a porton jelzi a modul, hogy készen van-e a művelettel. |

A negatív számokkal való szorzást úgy kezeljük, hogy megvizsgáljuk az operandusok legfelső bitjét. Amelyiknek 1-es (negatív), annak a kettes komplemensét képezzük – vagyis átalakítjuk pozitívra. A szorzást tehát minden esetben két pozitív számon végezzük el. Abban az esetben, ha az eredeti számok közül csak az egyik negatív volt, a végeredménynek szintén képezzük a kettes komplemensét, így az is negatív lesz.

# A diagram of a flowchart AI-generated content may be incorrect.Shift & add algoritmus [(1)](https://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-Mult.pdf)

Az algoritmus során két *n* bites számot (X, Y) szorzunk össze. A végeredményt *2n* biten ábrázoljuk. A szorzatregisztert (*A*) 0-ra inicializáljuk a szorzótényezőket betöltjük a *B* és *Q* regiszterekbe. Az *N* számláló értékét *n*-re állítjuk. Minden iteráció elején megvizsgáljuk, hogy *Q* legkisebb helyiértékű bitje (*Q0*) egyes-e. Amennyiben ez teljesül, hozzáadjuk az *A* regiszterhez a *B* értékét. Ezt követően *B*-t balra, *Q*-t jobbra shifteljük egy bittel, majd csökkentjük *N* értékét eggyel. Ha *N* értéke 0, akkor véget ér az algoritmus, minden más esetben folytatódik a ciklus.

1. A megszakításgenerálást a blokkdiagram 5. modulja végzi [↑](#footnote-ref-1)