Alkalmazásfejlesztés a PCDSP6 jelfeldolgozó kártyára

2009. November.

Dr. Gaál József

Alkalmazás fejlesztés a PCDSP6 kártyára

- Egyrészt
  - Funkcionális specifikáció
  - Elmélet, Algoritmus, Implementáció
  - Tesztelés, Dokumentálás

- Másrészt
  - Host programozása
  - DSP programozása
  - FPGA programozása

DSP programozás

- Kódolás
- Fordítás
- Linkelés
- Letöltés
- Futtatás
- Tesztelés, Vezérlés, Monitorozás

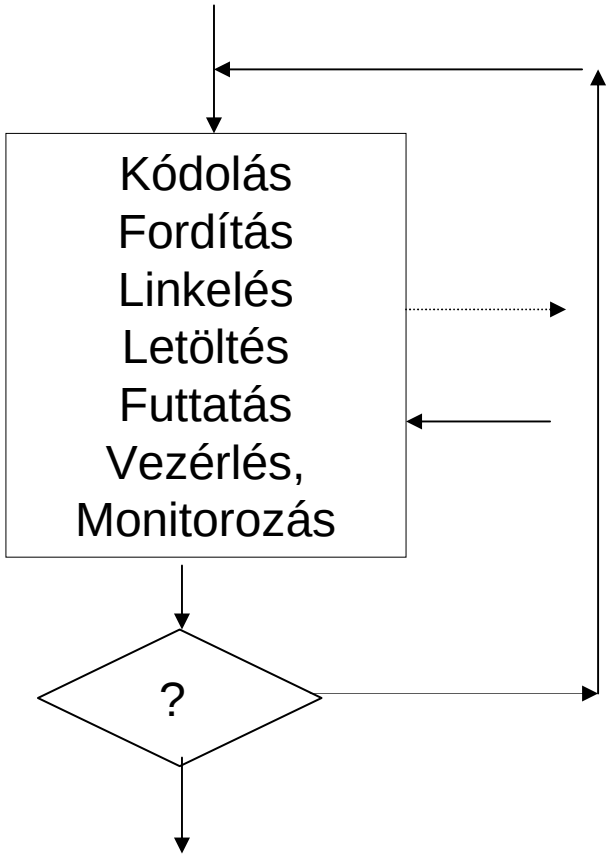
» Pontosabban szólva:

DSP programozás

- Különböző lépések, többszörösen visszacsatolt, iteratív folyamata

- Kulcskérdés:

  elméleti felkészültség és hatékony eszközhasználat



A DSP programozás főbb eszközei

- Code Composer Studio (Texas Instr.)
  - Kódolás
  - Fordítás (compilation, assembling)
  - Linkelés
  - Szimuláció, forrás szintű debuggolás
- Mon6000 (Relcom Kft.)
  - Letöltés
  - Futtatás
  - Vezérlés, Monitorozás

Szekciók (sections)

Több részből áll minden program: sections

Egyrészt: Vannak kód és adat részek, melyek **inicializálandóak** vagy nem :

- Kód: forrásnyelven megírandó, ezáltal inicializálandó
- Adatterületek
  - Inicializálandó vagy csak helyfoglalandó
  - Konstans vagy változtatható értékek

Másrészt: Vannak kód és adat részek, melyek fizikai, hardver okok miatt **különböző címen** helyezendőek el:

- Különböző funkcióhoz kötöttek:
  - Interrupt vektor, regiszterek, perifériák címei
- különböző típusú áramkörökbe valóak:
  - ROM, onchip RAM, external DRAM, stb.

Szekciók (sections)

A relokálható szekciók a fordítás alatt jönnek létre:

- egyrészt automatikusan,
- másrészt a forráskódban adott direktívák szerint.

A szekciók a linkelés során rendelődnek a különböző címtartományokhoz, és a mögöttük lévő különböző hardverekhez

- Egyrészt automatikusan,
- Másrészt a felhasználó által írt “linker kommander file”, *.cmd szerint.

Szekciók (sections)

A C fordító által automatikusan létrehozott (default)

- inicializált szekciók:
  - .text (program számláló)
  - .cinit
  - .const
  - .switch
- nem inicializált szekciók:
  - .bss (adat pointer DP .set B14)
  - .far (far pointer FP .set A15)
  - .stack (stack poinert SP .set B15)
  - .sysmem (heap memory)

Az assembler által automatikusan létrehozott (default)

- inicializált szekciók:
  - .text
  - .data
- nem inicializált szekciók:
  - .bss

Szekciók (sections)

A C fordító által létrehozott, felhasználó által definiált szekciók:

```
#pragma CODE_SECTION(fn, ”my_sect”)

int fn(int x)
{
      ……
      return x;
}

#pragma DATA_SECTION(bufferB, ”my_sect”)
char bufferA[512];
char bufferB[512];
```

Az assembler által létrehozott, felhasználó által generált

- inicializált szekciók:
  - **.sect “***section name***”**
- nem inicializált szekciók:
  - *symbol* **.usect “***section name***”,** *size in bytes* [**,** *alignment* [, *bank offset*]]

Szekciók (sections)

A linker commander file: Memóriák és szekciók ósszerendelése:

```
MEMORY
{
    InterrupServiceTable  (RX):   o = 00000000h    l = 00000200h        /* 512 bytes */
    OnChipRAM        (RWX):   o = 00000200h    l = 000FFE00h        /* 1M - 512 bytes */
    ExternalSDRAM    (RWX):   o = 80000000h    l = 08000000h        /* 128M bytes */
}
SECTIONS
{
    .ITvecs                   >          InterrupServiceTable
/*
    .stack                    >          OnChipRAM
    .far                      >          OnChipRAM
    .cinit                    >          OnChipRAM
*/
}
```

Az első DSP programunk

- A létrehozandó forrás file-ok:

- Interrupt vektor-tábla kitöltése:
  - vector.asm
- Az alkalmazás főprogramja:
  - pelda01.c
- Linker kommander File:
  - lnk_pelda.cmd

Az első DSP programunk

vector.asm :

```
FP  .set    A15
DP  .set    B14
SP  .set    B15

    .sect   .ITvecs

RESET_ISR:          ; Reset
    .ref    _c_int00
    MVKL .S2        (_c_int00), B0
    MVKH .S2        (_c_int00), B0
    B    .S2        B0
    NOP
    NOP
    NOP
    NOP
    NOP
```

```
ISR_RET .macro ret_poit
    B       ret_point
    NOP 7
    .endm

    ISR_RET         NRT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
    ISR_RET         IPT
```

Az első DSP programunk

pelda01.asm :

```
int loopCounter;

void idleFunction()
{
   loopCounter++;
}

void main()
{
  while(1)
    idleFunction();
}
```

lnk_pelda.cmd

```
MEMORY
{
 ISTable   (RX): o = 00000000h l=00000200h
 OnChipRAM(RWX): o = 00000200h l=000FFE00h
 ExtSDRAM (RWX): o = 80000000h l=08000000h
}
SECTIONS
{
  .ITvecs       > ISTable
}
```

Az első DSP programunk

pelda01.asm :

```
#pragma DATA_SECTION \
    (loopCounter, “adatok”);
int loopCounter;
void idleFunction()
{
   loopCounter++;
}

void main()
{
  while(1)    idleFunction();
}
```

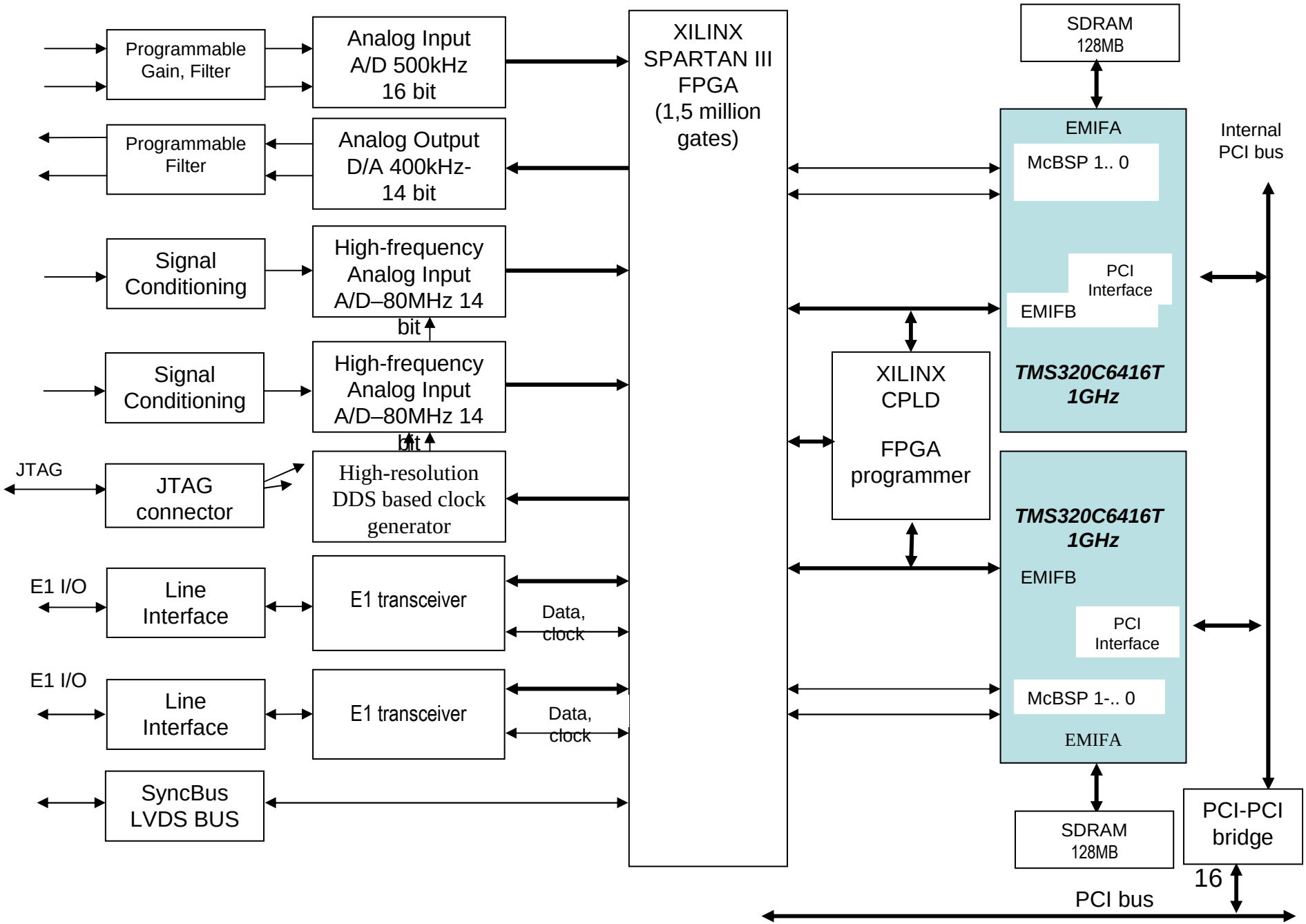
lnk_pelda.cmd

```
MEMORY
{
 ISTable   (RX): o = 00000000h l=00000200h
 OnChipRAM(RWX): o = 00000200h l=000EFE00h
 DATA_RAM (RWX): o = 000F0000h l=00010000h
 ExtSDRAM (RWX): o = 80000000h l=08000000h

}
SECTIONS
{
  .Itvecs    > ISTable
  adatok     >
}
```

Komponens szoftver

- RSL
- CSL
- DSP lib
- BSL
- Mintaprogramok


Programmable Gain, Filter
Analog Input A/D 500kHz 16 bit
Programmable Filter
Analog Output D/A 400kHz- 14 bit
Signal Conditioning
High-frequency Analog Input A/D–80MHz 14 bit
Signal Conditioning
High-frequency Analog Input A/D–80MHz 14 bit
JTAG
JTAG connector
High-resolution DDS based clock generator
E1 I/O
Line Interface
E1 transceiver
Data, clock
E1 I/O
Line Interface
E1 transceiver
Data, clock
SyncBus LVDS BUS
XILINX SPARTAN III FPGA (1,5 million gates)
XILINX CPLD
FPGA programmer
SDRAM 128MB
EMIFA
McBSP 1.. 0
PCI Interface
EMIFB
TMS320C6416T 1GHz
TMS320C6416T 1GHz
EMIFB
PCI Interface
McBSP 1-.. 0
EMIFA
SDRAM 128MB
Internal PCI bus
PCI-PCI bridge
16
PCI bus

• DSP memória térkép

DSP Address	Size	Description	Software
0000 0000 000F FFFF	1M	onchip RAM	
0010 0000 017F FFFF	23M	Reserved	
0180 0000 5FFF FFFF	1512M	onchip peripherials	CSL
6000 0000 6FFF FFFF	256M	external peripherials	BSL
7000 0000 7FFF FFFF	256M	Reserved	
8000 0000 87FF FFFF	128M	external **SDRAM**	
8800 0000 FFFF FFFF	1920M	Reserved	

Onchip perifériák 1.

DSPAddress	Size	Description	Software
0180 0000 0183 FFFF	256K	EMIFA registers	BSL
0184 0000 0187 FFFF	256K	CACHE registers	CSL
0188 0000 018B FFFF	256K	HPI registers	reserved
018C 0000 018F FFFF	256K	McBSP 0 registers	CSL
0190 0000 0193 FFFF	256K	McBSP 1 registers	CSL
0194 0000 0197 FFFF	256K	Timer 0 registers	CSL
0198 0000 019B FFFF	256K	Timer 1 registers	CSL
019C 0000 019F FFFF	256K	Interrupt selector registers	CSL

Onchip perifériák 2.

DSPAddress	Size	onchip peripherials (cont.1)	Software
01A0 0000 01A3 FFFF	256K	EDMA RAM and registers	CSL
01A4 0000 01A7 FFFF	256K	McBSP2 registers	CSL
01A8 0000 01AB FFFF	256K	EMIFB registers	BSL
01AC 0000 01AF FFFF	256K	Timer2 registers	CSL
01B0 0000 013F FFFF	256K	GPIO registers	CSL
01B4 0000 01B7 FFFF	256K	UTOPIA registers	reserved
01B8 0000 01BB FFFF	256K	TCP/VCP registers	CSL
01BC 0000 01BF FFFF	256K	Reserved	reserved
01C0 0000 01C3 FFFF	256K	PCI registers	CSL

Onchip perifériák 3.

DSPAddress	Size	onchip peripherials (cont.2)	Software
0200 0000 0200 0033	52	QDMA registers	?
0200 0034 2FFF FFFF	736M-52	Reserved	reserved
3000 0000 33FF FFFF	64M	McBSP0 Data	CSL
3400 0000 37FF FFFF	64M	McBSP1 Data	CSL
3800 0000 3BFF FFFF	64M	McBSP2 Data Not used	reserved
3C00 0000 3FFF FFFF	64M	Utopia queues Not used	reserved
4000 0000 4FFF FFFF	256M	Reserved	reserved
5000 0000 5FFF FFFF	256M	TCP/VCP	CSL

CSL

- **The chip support library (CSL)**
  - provides a C-language interface for configuring and controlling on-chip peripherals.
- **Benefits of the CSL**
  - Standard Protocol-to-Program Peripherals
  - Basic Resource Management (Open, Close)
  - Symbolic Peripheral Descriptions
- **CSL Architecture**
  - each peripheral is covered by a single API module.

CSL modules

Peripheral Module	Description	Include File	Module SupportSymbol
CACHE	Cache module	csl_cache.h	CACHE_SUPPORT
CHIP	Chip-specific module	csl_chip.h	CHIP_SUPPORT
CSL	Top-level module	csl.h NA	
DAT	Device independent data copy/fill module	csl_dat.h	DAT_SUPPORT
EDMA	Enhanced direct memory access module	csl_edma.h	EDMA_SUPPORT
EMIFA	External memory interface A module	csl_emifa.h	EMIFA_SUPPORT
EMIFB	External memory interface B module	csl_emifb.h	EMIFB_SUPPORT
GPIO	General-Purpose input/output module	csl_gpio.h	GPIO_SUPPORT
HPI	Host port interface module	csl_hpi.h	HPI_SUPPORT
I2C	Inter -Integrated circuit module	csl_i2c.h	I2C_SUPPORT
IRQ	Interrupt controller module	csl_irq.h	IRQ_SUPPORT
McASP	Multichannel audio serial port module	csl_mcasp.h	MCASP_SUPPORT
McBSP	Multichannel buffered serial port module	csl_mcbsp.h	MCBSP_SUPPORT
PCI	PCI module	csl_pci.h	PCI_SUPPORT
TCP	Turbo decoder coprocessor module	csl_tcp.h	TCP_SUPPORT
TIMER	Timer module	csl_timer.h	TIMER_SUPPORT
VCP	Viterbi decoder coprocessor module	csl_vcp.h	VCP_SUPPORT

CSL naming conventions

- Function PER_funcName()
- Variable PER_varName
- Macro PER_MACRO_NAME
- Typedef PER_Typename
- Function Argument funcArg
- Structure Member memberName

PER is the placeholder for the module name.

- The CSL macro and constant names are defined for each register and each field in CSL include files
- Generic convention

CSL data types

- **CSL Data Types**
  - Uint8 unsigned char
  - Uint16 unsigned short
  - Uint32 unsigned int
  - Uint40 unsigned long
  - Int8 char
  - Int16 short
  - Int32 int
  - Int40 long

CSL Functions

*handle=PER*_**open**(*channelNumber,[priority]flags*)
*PER*_**config**(*[handle,]*configStructure*)
*PER*_**configArgs**(*[handle,]regval_1,...regval_n)*
*PER*_**reset**(*[handle])*
*PER*_**close**(*handle*)

– Handle-based peripherals (Handles are required only for peripherals that have multiple channels or ports):
  - EDMA
  - GPIO
  - McBSP
  - TIMER
  - I2C

CSL Macros

PER	*peripherial*
REG	*register of peripherial*
FIELD	*field in register*
regval	*integer constant or variable or symbolic const*
fieldval	*integer constant or variable or symbolic const*
x	integer constant, integer variable.
sym	symbolic constant

- *PER_REG_***RMK**(*fieldval_n,...fieldval_0*)
- *PER_***RGET**(*REG*)
- *PER_***RSET**(*REG*,*regval*)
- *PER_***FMK** *(REG,*FIELD*,fieldval)*
- *PER_***FGET**(*REG*,*FIELD*)
- *PER_***FSET**(*REG,FIELD,fieldval*)
- *PER_REG_***ADDR**(*REG*)
- *PER_***FSETS** *(REG,*FIELD*,sym)*
- *PER_***FMKS** *(REG,*FIELD*,sym)*
- *PER_***ADDRH** (h,*REG*)
- *PER_***RGETH** (h**,***REG*)
- *PER_***RSETH** (h,*REG,*x)
- *PER_***FGETH** (h**,***REG,FIELD*)
- *PER_*F**SETH** (h**,***REG,FIELD,*x)
- *PER_***FSETSH** (h,*REG,FIELD,SYM*)

CSL Symbolic Constans

Generic CSL Symbolic Constants

Constant Values for Registers

– *PER_REG*_**DEFAULT**
  - Default value for a register; corresponds to the register value after a reset or to 0 if a reset has no effect.

Constant Values for Fields

– *PER_REG_FIELD_SYMVAL*
  - Symbolic constant to specify values for individual fields in the specified peripheral register. values.

– *PER_REG_FIELD*_**DEFAULT**
  - Default value for a field; corresponds to the field value after a reset or to 0 if a reset has no effect.

CSL

- The chip support library (CSL) has two layers:
  - the service layer and
  - the hardware abstraction layer, or HAL for short.

- The service layer contains
  - the API functions,
  - data types, and
  - constants as defined in the manual
- The HAL is made up of a set of header files that define an orthogonal set of
  - C macros and constants
  - which abstract registers and bitfields into symbols.

CSL

HAL Macro Type	Purpose
<PER>_ADDR	Register Address
<PER>_ADDRH	Register Address For Given Handle
<PER>_CRGET	Gets the Value of CPU Register
<PER>_CRSET	Sets the Value of CPU Register
<PER>_FGET	Field Get
<PER>_FGETA	Field Get Given Address
<PER>_FGETH	Field Get For Given Handle
<PER>_FMK	Field Make
<PER>_FMKS	Field Make Symbolically
<PER>_FSET	Field Set
<PER>_FSETA	Field Set Given Address
<PER>_FSETH	Field Set For Given Handle
<PER>_FSETS	Field Set Symbolically
<PER>_FSETSA	Field Set Symbolically For Given Address
<PER>_FSETSH	Field Set Symbolically For Given Handle
<PER>_RGET	Register Get
<PER>_RGETA	Register Get Given Address
<PER>_RGETH	Register Get For Given Handle
<PER>_RSET	Register Set
<PER>_RSETA	Register Set Given Address
<PER>_RSETH	Register Set For Given Handle
<PER>_<REG>_	DEFAULT Register Default Value
<PER>_<REG>_OF	Register Value Of …
<PER>_<REG>_RMK	Register Make
<PER>_<REG>_<FIELD>_DEFAULT	Field Default Value
<PER>_<REG>_<FIELD>_OF	Field Value Of …
<PER>_<REG>_<FIELD>_<SYM>	Field Symbolic Value

CSL: Timer

TIMER Configuration Structure

TIMER_Config	S	Structure used to set up a timer device

(a) Primary Functions

TIMER_close	F	Closes a previously opened timer device
TIMER_config	F	Configure timer using configuration structure
TIMER_configArgs		Sets up the timer using the register values passed in
TIMER_open	F	Opens a TIMER device for use
TIMER_pause	F	Pauses the timer
TIMER_reset	F	Resets the timer device associated to the handle
TIMER_resume	F	Resumes the timer after a pause
TIMER_start	F	Starts the timer device running

CSL: Timer

(b) Auxiliary Functions and Constants

TIMER_DEVICE_CNT	**C**	**A compile time constant; number of timer devices present**
TIMER_getBiosHandle	**F**	**Returns the timer handle of the timer used by BIOS**
TIMER_getConfig	**F**	**Reads the current Timer configuration values**
TIMER_getCount	**F**	**Returns the current timer count value**
TIMER_getDatIn	**F**	**Reads the value of the TINP pin**
TIMER_getEventId	**F**	**Obtains the event ID for the timer device**
TIMER_getPeriod	**F**	**Returns the period of the timer device**
TIMER_getTStat	**F**	**Reads the timer status; value of timer output**
TIMER_resetAll	**F**	**Resets all timer devices**
TIMER_setCount	**F**	**Sets the count value of the timer**
TIMER_setDatOut	**F**	**Sets the data output value**
TIMER_setPeriod	**F**	**Sets the timer period**
TIMER_SUPPORT	**C**	**A compile time constant whose value is 1 if the device supports the TIMER module**

CSL: Timer

TIMER Macros that Access Registers and Fields

Macro	Description/Purpose
TIMER_ADDR(<REG>)	Register address
TIMER_RGET(<REG>)	Returns the value in the peripheral register
TIMER_RSET(<REG>,x)	Register set
TIMER_FGET(<REG>,<FIELD>)	Returns the value of the specified field in the peripheral register
TIMER_FSET(<REG>,<FIELD>,fieldval)	Writes *fieldval* to the specified field in the peripheral register
TIMER_FSETS(<REG>,<FIELD>,<SYM>)	Writes the symbol value to the specified field in the peripheral
TIMER_RGETA(addr,<REG>)	Gets register for a given address
TIMER_RSETA(addr,<REG>,x)	Sets register for a given address
TIMER_FGETA(addr,<REG>,<FIELD>)	Gets field for a given address
TIMER_FSETA(addr,<REG>,<FIELD>, fieldval)	Sets field for a given address

CSL: Timer

TIMER Macros that Access Registers and Fields

Macro Description/Purpose

TIMER_FSETSA(addr,<REG>,<FIELD>,<SYM>)
Sets field symbolically for a given address

TIMER_ADDRH(h,<REG>)
Returns the address of a memory-mapped register for a given handle

TIMER_RGETH(h,<REG>)
Returns the value of a register for a given handle

TIMER_RSETH(h,<REG>,x)
Sets the register value to x for a given handle

TIMER_FGETH(h,<REG>,<FIELD>)
Returns the value of the field for a given handle

TIMER_FSETH(h,<REG>,<FIELD>,fieldval)
Sets the field value to x for a given handle

CSL: Timer

TIMER Macros that Construct Register and Field Values

Macro	**Description/Purpose**
TIMER_<REG>_DEFAULT	Register default value
TIMER_<REG>_RMK()	Register make
TIMER_<REG>_OF()	Register value of ...
TIMER_<REG>_<FIELD>_DEFAULT	Field default value
TIMER_FMK()	Field make
TIMER_FMKS()	Field make symbolically
TIMER_<REG>_<FIELD>_OF()	Field value of ...
TIMER_<REG>_<FIELD>_<SYM>	Field symbolic value

CSL példa: Timer

```
#include <csl.h>
#include <csl_timer.h>
TIMER_Handle      hTimer0, hTimer1;
Uint32 prd0=0x11111111, tim0=0x33333333;
int loop_counter=0x12345678;
int tint0_counter=0x10000000;
int tint1_counter=0x20000000;
interrupt void ISR_TINT0() {
     tint0_counter++;
}
interrupt void ISR_TINT1() {
     tint1_counter++;
}
void idleFunction(){
     loop_counter++;
     prd0 = TIMER_getPeriod(hTimer0);
     tim0 = TIMER_getCount(hTimer0);
}
```

CSL példa: Timer

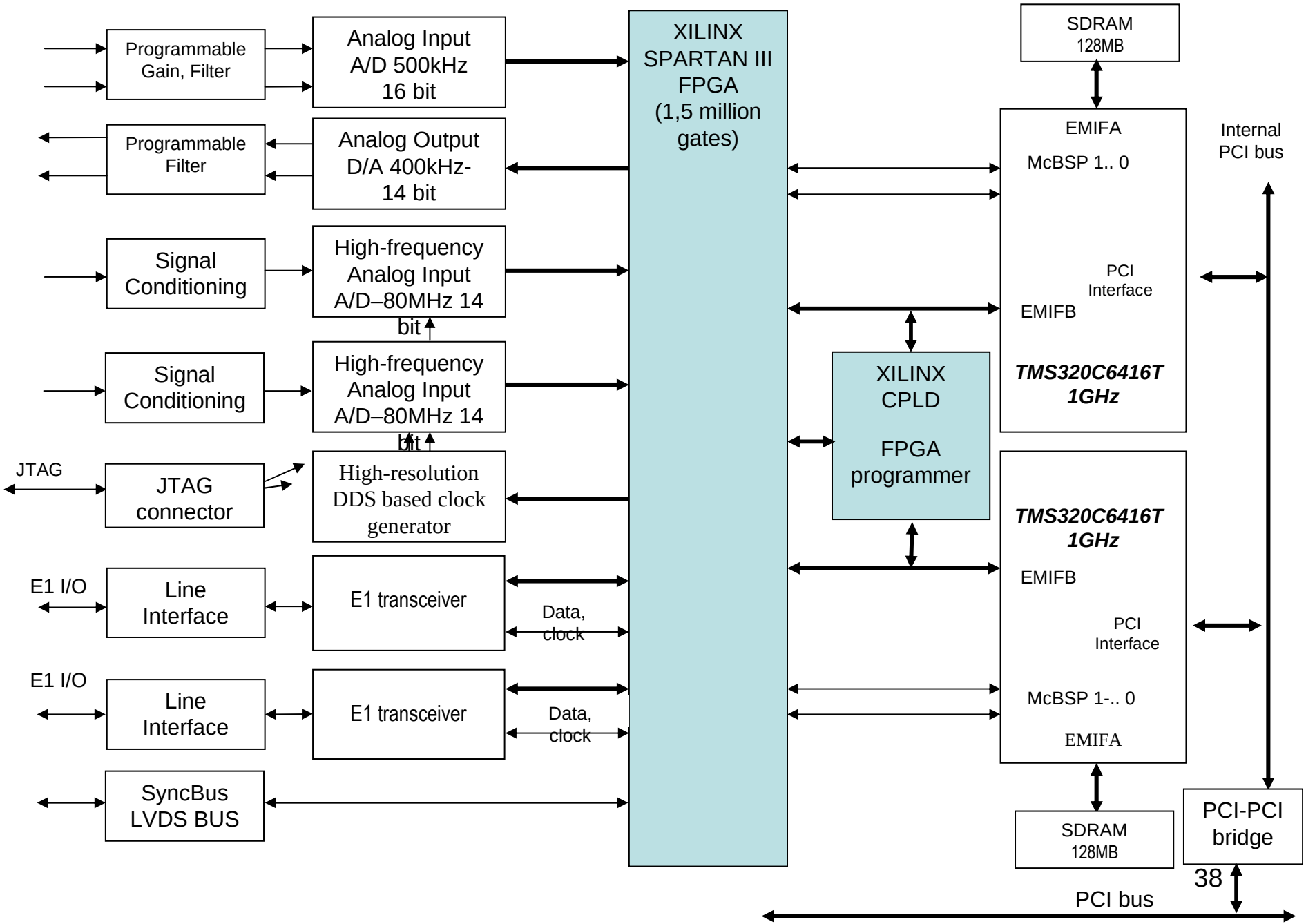
```
void main(){
    IER = 0x0003;   // enable Reset and NMI only
    ICR = 0xFFFF;  // clear all pending ITs
    CSR |= 1;                       // enable Global Interrupt Enable (GIE)
    CSL_init();

    hTimer0 = TIMER_open(TIMER_DEV0, TIMER_OPEN_RESET);

    TIMER_configArgs( hTimer0,
            TIMER_CTL_OF( 0x00000211 ),
            TIMER_PRD_OF( 125000000 ),      // = 1 Hz (pulse) @ 1 GHz CPU
            TIMER_CNT_OF( 0x00000000 )
    );
    IRQ_enable( IRQ_EVT_TINT0 );
    TIMER_start( hTimer0 );
    while ( 1) idleFunction();  // allandoan fut
}
```

BSL: Board Support Library

DSP Address	Size	Description	Software
0000 0000 000F FFFF	1M	onchip RAM	
0010 0000 017F FFFF	23M	Reserved	
0180 0000 5FFF FFFF	1512M	onchip peripherials	CSL
6000 0000 6FFF FFFF	256M	external peripherials	BSL
7000 0000 7FFF FFFF	256M	Reserved	
8000 0000 87FF FFFF	128M	external **SDRAM**	
8800 0000 FFFF FFFF	1920M	Reserved	


Programmable Gain, Filter
Analog Input A/D 500kHz 16 bit
Programmable Filter
Analog Output D/A 400kHz- 14 bit
Signal Conditioning
High-frequency Analog Input A/D–80MHz 14 bit
Signal Conditioning
High-frequency Analog Input A/D–80MHz 14 bit
JTAG
JTAG connector
High-resolution DDS based clock generator
E1 I/O
Line Interface
E1 transceiver
Data, clock
E1 I/O
Line Interface
E1 transceiver
Data, clock
SyncBus LVDS BUS
XILINX SPARTAN III FPGA (1,5 million gates)
XILINX CPLD
FPGA programmer
SDRAM 128MB
EMIFA
McBSP 1.. 0
PCI Interface
EMIFB
TMS320C6416T 1GHz
TMS320C6416T 1GHz
EMIFB
PCI Interface
McBSP 1-.. 0
EMIFA
SDRAM 128MB
Internal PCI bus
PCI-PCI bridge
38
PCI bus

BSL: Board Support Library

- CSL mintájára
  - HAL: Hardware Abstraction Layer
  - Macrók, konstansok
- Eszközök: CPLD, FPGA, egyéb
  - Perifériák header file-ok
    - Regiszterek
      - Bit-mezők

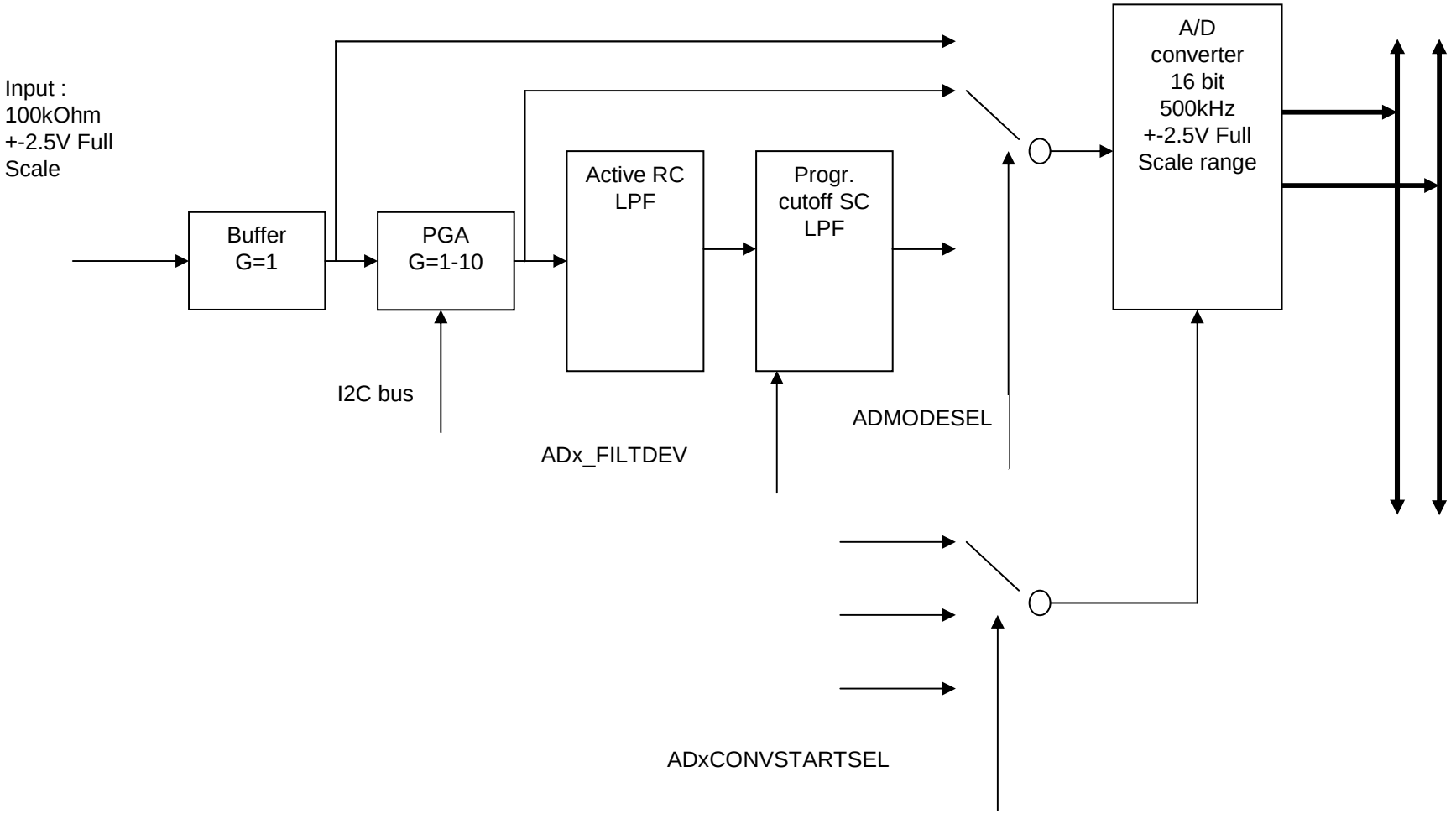
BSL: CPLD-ben realizált perifériák

- AOUT Analog Output Control bsl_aouthal.h
  - Szűrt vagy szűretlen A/D kimenet
- BBAIO Base-band analog input/outpu bsl_bbaiohal.h
  - Adatok irása, olvasása, (aszinkron) start konverzió parancsok
- BDAC Analog Output Control bsl_bdachal.h
  - Szűrt vagy szűretlen A/D kimenet
- FPGA FPGA Program Control bsl_fpgahal.h
  - PROG,INIT,CS,RDWR,DATA,BUSZ,DONE regiszterek
- MISC Misc. Control/Status bsl_mischal.h
  - DSPID, I2C bus, LED, DORBELL, NMICMD, EI7SRC

BSL: FPGA-ban realizált perifériák

- UBADC Base-band Analog-to-Digital Control bsl_ubadchal.h
  - Szűrők prog., trigger select, jel-kondicionálás (erősités, szűrés)
- UBDAC Base-band Digital-to-Analog Control bsl_ubdachal.h
  - DSP1vagy2, szűrők prog., trigger select
- UDDS DDS Chip Control bsl_uddshal.h
  - DDS W/R, enable, chipselect, clock, data
- UFRMR Framer Control bsl_ufrmrhal.h
  - DSP1vagy2, Clock select
- UHADC High-speed Analog-to-Digital Control bsl_uhadchal.h
  - Clock select, FIFO status
- UINT Interrupt Control bsl_uinthal.h
  - INT4(56)Source, INT4(56)Send
- UMISC Miscellaneous registers bsl_umischal.h
  - DSP clock source select
- UMLVDSB MMLVDSBUS bsl_umlvdsbhal.h
- USBUS Syncbus bsl_usbushal.h

Példa: BBAIO_test



Példa: BBAIO_test