

Application Note



Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

UTIA EdkDSP Platform PB2 Firmware Programming Interface

Roman Bartosinski

bartosr@utia.cas.cz

Revision

Revision	Date	Author	Description
0	11.3.2011	Bartosinski	document creation

Contents

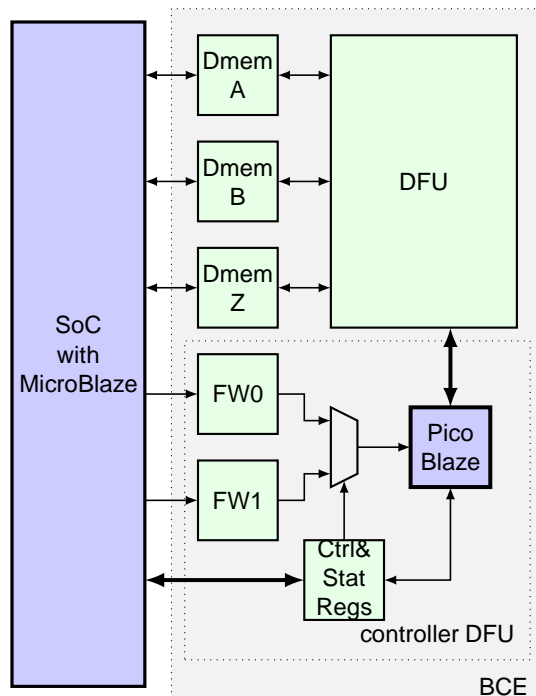
1	Introduction	1
2	Content of the programming interface	1
3	How to use the programming interface	2
3.1	Include API header file	2
3.2	Use functions from API	2
3.3	Build firmware binary	3
3.4	Example of firmware with the API	3
4	Application Programming Interface	4
4.1	enum dfu_fp01_operations	4
4.2	enum dfu_fp01_memories	5
4.3	function pb2mb_read_data	6
4.4	function pb2mb_write_data	6
4.5	function pb2mb_eoc	7
4.6	function pb2mb_req_reset	7
4.7	function pb2mb_reset	8
4.8	function pb2dfu_wait4hw	8
4.9	function pb2dfu_start_op	9
4.10	function pb2dfu_restart_op	9
4.11	function pb2dfu_set_cnt	10
4.12	function pb2dfu_set_addr	10
4.13	function pb2dfu_set_bank	11
4.14	function pb2dfu_set_fulladdr	11
4.15	function pb2dfu_set_inc	12
4.16	function pb2dfu_set_restart_addr	13

Acknowledgement

The research leading to these result has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100230 and from the MSMT 7H10001.

1 Introduction

The application programming interface (API), described in this document, is one part of a software development kit for hardware accelerators based on EdkDSP platform (EdkDSP SDK). The API version 1.0 for accelerator's microcontroller is described in this document. In this case, a PicoBlaze (PB) is used as a microcontroller and a 32bit processor MicroBlaze (MB) is used as the host CPU.



The API defines an interfaces between the microcontroller and data flow unit (PB2DFU) and between the accelerator and the host CPU from the side of the accelerator (PB2MB). It offers functions for communication with the host CPU and functions to parameterize and control basic computing operations in hardware.

The EdkDSP platform is described in the deliverable D2.1 in SMECY project [1].

The document is organized as follows: The first part describes how to use the API in user firmwares and the second part contains description of all functions and constants in the API which is generated from the source code.

2 Content of the programming interface

The firmware programming interface is distributed as one header file for C compiler and library as one object file. The API closely depends with the hardware data flow unit, therefore the API can be different for each accelerator and therefore the files of the interface haven't universal names. In this case, the API contains two files `dfu_fp01_1x1.h` and `dfu_fp01_1x1.psmo`.

The programming interface is prepared for using with tools from EdkDSP SDK which are distributed in package `pb-toolchain`. The toolchain compiles source codes in limited ANSI C to binary firmware for the PicoBlaze microcontroller.

Version of the current PB C compiler doesn't optimize output code and therefore the PB2 API contains temporary specific versions of functions in files `dfu_fp01_1x1_fir`. These functions are designed to save space in accelerator's firmware memories. This temporary version of the API is used only in a large example for computation FIR and LMS filters in the accelerators.

The PB2 API is distributed in the following directory structure

```

.
|-- api
    |-- 00-pb-firmware
        |-- doc
        |   |-- pb2_api.pdf
        |-- dfu_fp01_1x1.h
        |-- dfu_fp01_1x1.psmo
        |-- dfu_fp01_1x1_fir.h
        |-- dfu_fp01_1x1_fir.psmo

```

The following directory structure is considered in all examples in this document. All codes for accelerator's firmware are in directories 00-pb-firmware.

```

.
|-- api
|   |-- 00-pb-firmware
|   |-- 01-mb-standalone
|-- example1
|   |-- 00-pb-firmware
|   |-- 01-mb-standalone

```

3 How to use the programming interface

Usage of the API is simple and can be described with the following steps

- include API header file to the C source code,
- use functions from API in the source code,
- Build firmware binary from source codes and the API library.

3.1 Include API header file

Functions in the programming interface are declared in the `dfu_fp01_1x1.h` header file. It must be included in the source code with the preprocessor directive `#include`.

```
#include "../api/00-pb-firmware/dfu_fp01_1x1.h"
```

3.2 Use functions from API

All functions and constants from the API can be used after including API header file. The firmware must perform communication between accelerator and host CPU. Functions with prefixes `'pb2mb_'` and `'mb2pb_'` in the API are designed for this purpose. Using of these functions defines the communication protocol.

Functions with prefix `'pb2dfu'` serve to control data flow unit in the accelerator. These functions set parameters of the next operation (`pb2dfu_set_`), launch the operation (`pb2dfu_start_op`, `pb2dfu_restart_op`) and wait for finishing the operation (`pb2dfu_wait4hw`). Parameters of the next operation can be set before the previous operation is done, but it can be launched not before the previous operation is done. This feature allows to launch operation with minimal delay. The data flow unit remember the last settings and therefore only changed parameters can be set for the next operation.

3.3 Build firmware binary

Firmwares in C source codes must be compiled with the EdkDSP SDK toolchain distributed in the package pb-toolchain. The simplest way how to compile and build firmware binary is in the following listing. In the listing, we consider that the toolchain is installed and we compile the example1 in directory ./example1/00-pb-firmware with directory structure as shown in section 2.

```
pbcc ../../api/00-pb-firmware/dfu_fp01_1x1.psmo fw_ex1.c -o fw_ex1.h
```

In this example, the output binary has form of C header file with array of binary codes of firmware. Such file will be used to compile application for the host CPU as described in the documentation of the WAL API [2].

3.4 Example of firmware with the API

This part shows simple example of firmware, which waits for operation required by the host CPU and if the operation is get_capability the operation is starts in the accelerator. When the operation is finished the controller sends messages End of Computation, Request for Reset and Reseted to the host CPU.

```
/* include library for FP01 DFU */
#include "../../api/00-pb-firmware/dfu_fp01_1x1.h"

void main()
{
    unsigned char op;

    /* waiting for data from MB (the first byte is a required operation) */
    op = mb2pb_read_data();

    if (op==DFU_OP_VVER) { /* FW support only this one operation */
        /* start DFU with required operation */
        pb2dfu_start_op(DFU_OP_VVER, 1);
    }
    /* waiting for operation is done */
    pb2dfu_wait4hw();
    /* send message 'end of computation' to the host CPU (MB) */
    pb2mb_eoc('.');
    /* send message 'require reset' to the host CPU (MB) - communication uses handshaking */
    pb2mb_req_reset('.');
    /* reset PB and set status register for the host CPU (MB) */
    pb2mb_reset();
    /* endless loop - waiting to hardware reset of PB */
    while (1)
    ;
}
```

4 Application Programming Interface

4.1 enum dfu_fp01_operations

Purpose

enum dfu_fp01_operations - codes of operations supported by accelerator DFU_FP01

Synopsis

```
enum dfu_fp01_operations {  
    DFU_OP_VVER,  
    DFU_OP_VZ2A,  
    DFU_OP_VB2A,  
    DFU_OP_VZ2B,  
    DFU_OP_VA2B,  
    DFU_OP_VADD,  
    DFU_OP_VADD_BZ2A,  
    DFU_OP_VADD_AZ2B,  
    DFU_OP_VSUB,  
    DFU_OP_VSUB_BZ2A,  
    DFU_OP_VSUB_AZ2B,  
    DFU_OP_VMULT,  
    DFU_OP_VMULT_BZ2A,  
    DFU_OP_VMULT_AZ2B,  
    DFU_OP_VPROD,  
    DFU_OP_VMAC,  
    DFU_OP_VMSUBAC  
};
```

Constants

<i>DFU_OP_VVER</i>	ask about HW version
<i>DFU_OP_VZ2A</i>	copy vector $a[i] = z[j]$
<i>DFU_OP_VB2A</i>	copy vector $a[i] = b[j]$
<i>DFU_OP_VZ2B</i>	copy vector $b[i] = z[j]$
<i>DFU_OP_VA2B</i>	copy vector $b[i] = a[j]$
<i>DFU_OP_VADD</i>	add vectors $z[i] = a[j] + b[k]$
<i>DFU_OP_VADD_BZ2A</i>	add vectors $a[i] = b[j] + z[k]$
<i>DFU_OP_VADD_AZ2B</i>	add vectors $b[i] = a[j] + z[k]$
<i>DFU_OP_VSUB</i>	sub vectors $z[i] = a[j] - b[k]$
<i>DFU_OP_VSUB_BZ2A</i>	sub vectors $a[i] = b[j] + z[k]$
<i>DFU_OP_VSUB_AZ2B</i>	sub vectors $b[i] = a[j] + z[k]$
<i>DFU_OP_VMULT</i>	mult vectors $z[i] = a[j] * b[k]$
<i>DFU_OP_VMULT_BZ2A</i>	mult vectors $a[i] = b[j] * z[k]$
<i>DFU_OP_VMULT_AZ2B</i>	mult vectors $b[i] = a[j] * z[k]$
<i>DFU_OP_VPROD</i>	vector product $z = a[i..i+nn] * b[i..i+nn]$
<i>DFU_OP_VMAC</i>	vector MAC $z[i] = z[i] + a[j] * b[k] \ 1..13$
<i>DFU_OP_VMSUBAC</i>	vector MSUBAC $z[i] = z[i] - a[j] * b[k] \ 1..13$

Description

These codes are used with functions `pb2dfu_start_op` and `pd2dfu_restart_op` to select operation performed in the accelerator.

4.2 enum dfu_fp01_memories

Purpose

`enum dfu_fp01_memories` - identifiers of data memories in DFU_FP01 accelerator

Synopsis

```
enum dfu_fp01_memories {  
    DFU_MEM_A,  
    DFU_MEM_B,  
    DFU_MEM_Z  
};
```

Constants

<i>DFU_MEM_A</i>	target memory is data memory A
<i>DFU_MEM_B</i>	target memory is data memory B
<i>DFU_MEM_Z</i>	target memory is data memory Z

Description

The identifiers are used with functions `pb2dfu_set_...` to select target data memory.

4.3 function `pb2mb_read_data`

Purpose

`pb2mb_read_data` - read (blocking) data from a host CPU (MB)

Synopsis

```
unsigned char pb2mb_read_data ()
```

Description

The function communicates with a host CPU (MB) and waits for one data byte which is acknowledged.

Return Value

The function returns value of the received data byte.

4.4 function `pb2mb_write_data`

Purpose

`pb2mb_write_data` - send (blocking) data to a host CPU (MB)

Synopsis

```
void pb2mb_write_data (unsigned char data)
```


Arguments

data

one byte which will be send to MB

Description

The function sends one data byte and wait for acknowledgement.

Return Value

The function doesn't return any value.

4.5 function pb2mb_eoc

Purpose

pb2mb_eoc - send (blocking) a message 'End Of Computation' to a host CPU (MB)

Synopsis

```
void pb2mb_eoc (unsigned char data)
```

Arguments

data

one data byte which will be send to MB with the EOC message

Description

The function sends message End of Computation (EOC) with an optional data byte to a host CPU (MB). Then it waits for acknowledgement from the host CPU. The message should be send anytime the accelerator wants to synchronize computation with a host CPU. Usually, the message is sent after the entire computation.

Return Value

The function doesn't return any value.

4.6 function pb2mb_req_reset

Purpose

pb2mb_req_reset - send (blocking) a message 'Request for Reset' to a host CPU (MB)

Synopsis

```
void pb2mb_req_reset (unsigned char data)
```

Arguments

data

one data byte which will be send to MB with the RR message

Description

The function sends message 'Request for Reset' (RR) with an optional data byte to a host CPU (MB). Then it waits for acknowledgement from the host CPU. The message should be send before resetting controller to inform the host CPU that the accelerator's controller will be reset to the initial state.

Return Value

The function doesn't return any value.

4.7 function pb2mb_reset

Purpose

pb2mb_reset - reset accelerator itself

Synopsis

```
void pb2mb_reset ()
```

Description

The function sets accelerator's controller to the initial state and set the accelerator's status bit.

Return Value

The function doesn't return any value.

4.8 function pb2dfu_wait4hw

Purpose

pb2dfu_wait4hw - PB will wait for end of computation

Synopsis

```
void pb2dfu_wait4hw ()
```

Description

The function waits for finishing computation in the accelerator. The function should be called before subsequent run of the next operation. The next operation can be prepared before the waiting to speed up the entire computation.

Return Value

The function doesn't return any value.

4.9 function pb2dfu_start_op

Purpose

pb2dfu_start_op - start operation in DFU with specified length of data vectors

Synopsis

```
void pb2dfu_start_op (unsigned char op, unsigned char cnt)
```

Arguments

<i>op</i>	DFU operation (constants DFU_OP_XXX)
<i>cnt</i>	length of input data vectors

Description

The function covers two functions (pb2dfu_set_cnt and pb2dfu_restart_op).

Return Value

The function doesn't return any value.

4.10 function pb2dfu_restart_op

Purpose

pb2dfu_restart_op - start operation in DFU

Synopsis

```
void pb2dfu_restart_op (unsigned char op)
```

Arguments

<i>op</i>	DFU operation (constants DFU_OP_XXX)
-----------	--------------------------------------

Description

All parameters of the operation must be set before this function. All parameters are registered and so only changed parameters from previous operations must be set. On the other hand, the operation must be always set because the function starts a required operation in the DFU.

Return Value

The function doesn't return any value.

4.11 function `pb2dfu_set_cnt`

Purpose

`pb2dfu_set_cnt` - set length of input data vectors for the next operation

Synopsis

```
void pb2dfu_set_cnt (unsigned char cnt)
```

Arguments

<i>cnt</i>	length of input data vectors
------------	------------------------------

Description

The function sets length of the input data vectors. The simple operations (as VADD, VMULT) will be performed *cnt*-times as one pipelined operation.

Return Value

The function doesn't return any value.

4.12 function `pb2dfu_set_addr`

Purpose

`pb2dfu_set_addr` - set address of the first element in the vector

Synopsis

```
void pb2dfu_set_addr (unsigned char mem, unsigned char addr)
```

Arguments

<i>mem</i>	select data memory (constant DFU_MEM_XXX)
<i>addr</i>	address of the first element in vector saved in memory <i>mem</i> in the current bank

Description

The function sets lower part of the initial pointer (index of the first element of vector in the specified memory bank) to `addr` for memory `mem` and bank which is set with function `pb2dfu_set_bank`. All data memories are organized into memory banks with 256 elements. The value will be used with the next operation.

Return Value

The function doesn't return any value.

4.13 function `pb2dfu_set_bank`

Purpose

`pb2dfu_set_bank` - select bank for specified memory

Synopsis

```
void pb2dfu_set_bank (unsigned char mem, unsigned char bank)
```

Arguments

<i>mem</i>	select data memory (constant <code>DFU_MEM_XXX</code>)
<i>bank</i>	bank which will be used for the next operation

Description

The function sets higher part of the initial pointer (bank of memory `mem` used for the next operation) to `bank` for memory `mem`. Lower part of the initial pointer can be set with function `pb2dfu_set_addr`. The value will be used with the next operation.

Return Value

The function doesn't return any value.

4.14 function `pb2dfu_set_fulladdr`

Purpose

`pb2dfu_set_fulladdr` - set full address (bank and offset) of the first element in the vector

Synopsis

```
void pb2dfu_set_fulladdr (unsigned char mem, unsigned char bank, unsigned char addr)
```

Arguments

<i>mem</i>	select data memory (constant DFU_MEM_XXX)
<i>bank</i>	bank of memory <i>mem</i> which will be used for the next operation
<i>addr</i>	address of the first element in vector saved in memory <i>mem</i> in the selected bank

Description

The function sets full address of the initial pointer (the first element of a vector) in memory *mem*. Each full address consists of bank and address in the bank which can be also set separately with functions `pb2dfu_set_bank` and `pb2dfu_set_addr`. The address will be used with the next operation.

Return Value

The function doesn't return any value.

4.15 function `pb2dfu_set_inc`

Purpose

`pb2dfu_set_inc` - set increment of offset for vector in selected memory

Synopsis

```
void pb2dfu_set_inc (unsigned char mem, unsigned short inc)
```

Arguments

<i>mem</i>	select data memory (constant DFU_MEM_XXX)
<i>inc</i>	increment between two elements of vector in the memory <i>mem</i>

Description

The function sets address increment to *inc* for vector from memory *mem*. Pointer to memory *mem* is increment about *inc* before processing the next vector element. The value will be used with the next operation.

Return Value

The function doesn't return any value.

4.16 function `pb2dfu_set_restart_addr`

Purpose

`pb2dfu_set_restart_addr` - set restart address for using vector elements in loop

Synopsis

```
void pb2dfu_set_restart_addr (unsigned char mem, unsigned char addr)
```

Arguments

<i>mem</i>	select data memory (constant <code>DFU_MEM_XXX</code>)
<i>addr</i>	restart address (lower part of the full address)

Description

The restart address `addr` is used if elements of the vector in memory `mem` are used in a loop. When the pointer to the next element overflow to the other bank it is set (lower part of the full address) to the restart address `addr`. The value will be used with the next operation.

Return Value

The function doesn't return any value.

References

- [1] J. Kadlec and all, "D2.1 - preliminary report on platform dependent parameters and optimizations," August 2010. SMECY project deliverable D2.1.
- [2] R. Bartosinski, "UTIA EdkDSP Platform, WAL - Worker Abstraction Layer," March 2011. SMECY project deliverable D2.2.