

Blackfin BF533 EZ-KIT Control

JITK -- Putting the O into I/O

Activating a FLASH memory "output line"
Part 2

Agenda

- Processors need to send out control signals (high / low 1 / 0 true / false)
 - General purpose input / output GPIO on processor chip (16)
 - FLASH memory chip has additional I/O ports connected to Ez-Lite KIT LED's
- Making the FLASH memory I/O port control the Ez-KIT LED's
- The new Blackfin assembly language instructions needed

Blackfin BF533 I/O

2

Blackfin I/O pins -- REVIEW

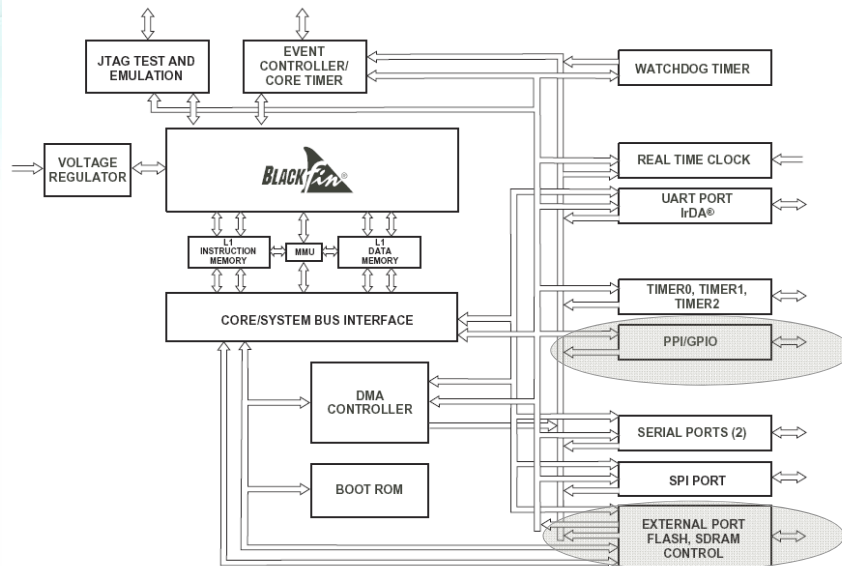


Figure 1-1. Processor Block Diagram

3

Radio controlled car

"IN PRINCIPLE", we could

- Connect LED1 control signal to turn right signal line of radio transmitter
- Connect LED2 control signal to forward signal line of radio transmitter
- Connect LED3 control signal to left signal line of radio transmitter

"IN PRINCIPLE" means – we might start off this way while we initially explore ideas to control the car.

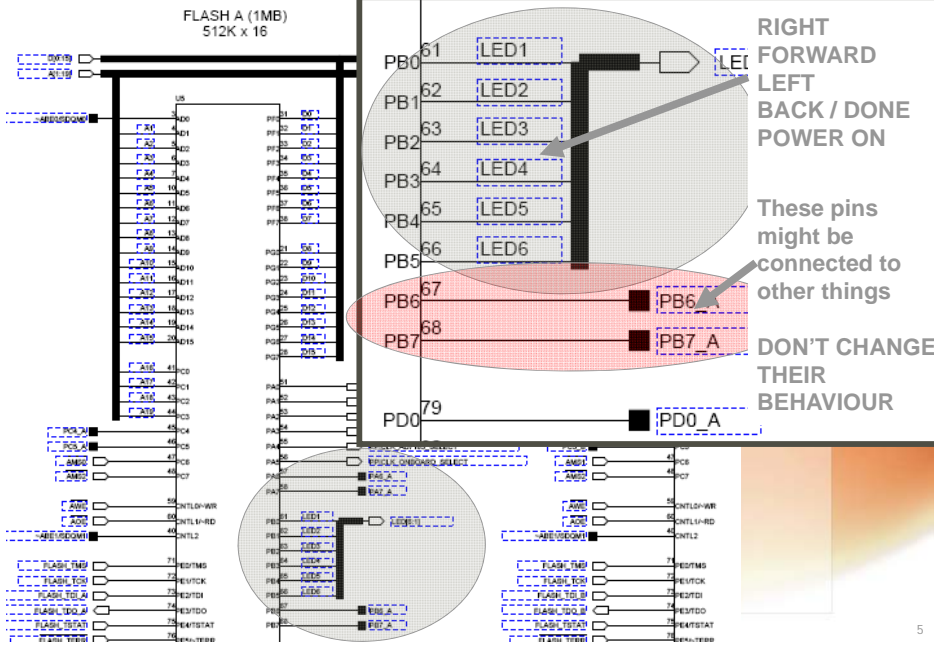
However we may (or may not) finish the project a different way.

- In actual fact we will use both PF1, PF5, PF6, PF7 as output to control car during the labs. **IN PRINCIPLE** – During Lab 4 we could use SPI interface so we can control car and put out messages to operator on LCD screen. (Same SPI as used during TV flashing ship lab.)

Blackfin BF533 I/O

4

LEDs connected to memory port



Activating LEDs -- REVIEW

- Get the FLASH to work correctly
 - Performed by *InitFlashCPP()*
- Get the Port to work correctly as output for pins PB5 → PB0, leaving other pins unchanged in behaviour
 - Performed by *InitFlashPortCPP()*
- Write the value we want to LEDs
 - *WriteFlashLEDASM(int value)*
 - *WriteFlashLEDCPP(int value)*
- Read back the value the LEDs show
 - *int ReadFlashLEDASM(void)*
 - *int ReadFlashLEDCPP(void)*

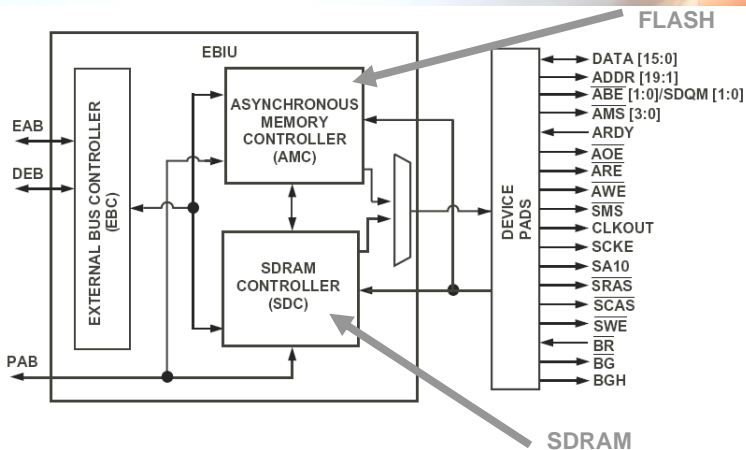
Blackfin BF533 I/O

CHANGE INTERFACING FROM ASM TO CPP CODE

“EBIU” External Bus Interface Unit -- REVIEW

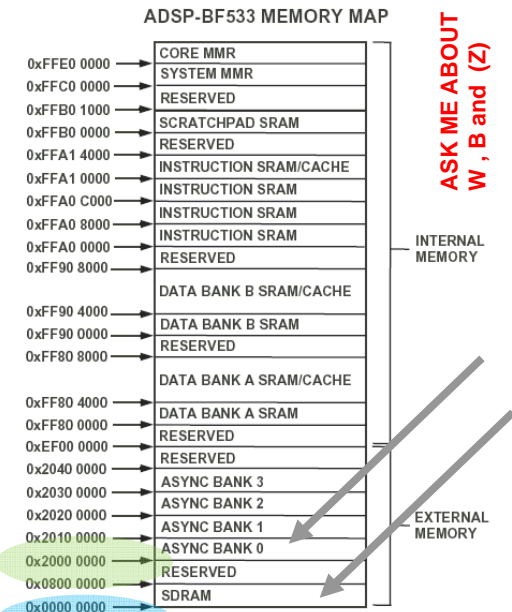
How does EBIU “know” whether to execute your code by writing the data to FLASH (LEDs live there) or SDRAM memory (Large arrays live there)?

- JITK -- Just-In-Time-Knowledge – WAIN rather than WAIL



ANSWER -- Blackfin Memory Map

- **McVASH control logic ideas again!**
- LDF file controlled
- If P0 (pointer register) is set to address 0x20001000 then
 - $R0 = W[P0] (Z);$ reads a 32-bit value from FLASH BANK 0
- If R0 is 6 and P0 is 0x0000 1000 then
 - $B[P0] = R0;$ places an 8-bit value into SDRAM memory



FLASH registers -- REVIEW

How does Blackfin “match” itself for fastest FLASH operation

- Depends on which FLASH is used in the EZ-Lite KIT from a specific manufacturer!

EBIU Programming Model

This section describes the programming model of the EBIU. This model is based on system memory-mapped registers used to program the EBIU.

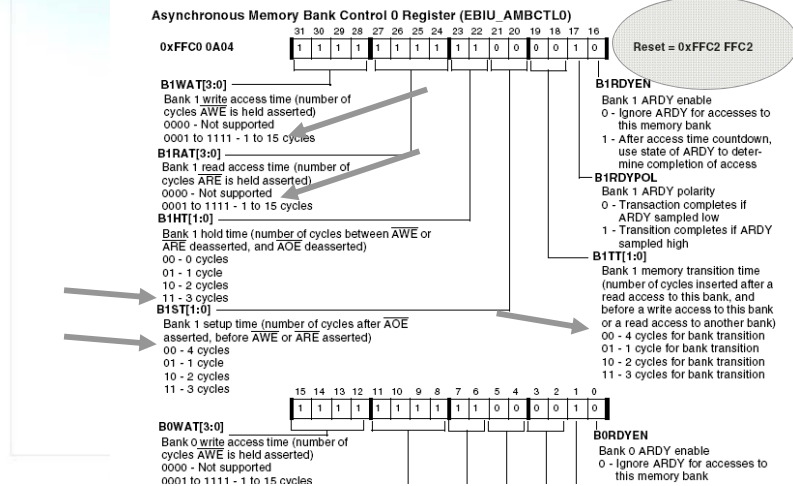
There are six control registers and one status register in the EBIU. They are:

- Asynchronous Memory Global Control register (EBIU_AMGCTL)
- Asynchronous Memory Bank Control 0 register (EBIU_AMBCTL0)
- Asynchronous Memory Bank Control 1 register (EBIU_AMBCTL1)
- SDRAM Memory Global Control register (EBIU_SDGCTL)

Bank control register -- REVIEW

Reset value will probably work “as is” but not efficient – “slow reads”

- Efficiency not normally a problem – if op not done often



General Control Register -- REVIEW

Reset value leaves “CLKOUT” disabled – is that important?

Asynchronous Memory Global Control Register (EBIU_AMGCTL)



Figure 17-3. Asynchronous Memory Global Control Register

InitFlashCPP() -- REVIEW

- Get the FLASH to work correctly
- May be “many” processes running on the Blackfin. All these processes may want to use InitFlashCPP()

```
InitFlashCPP() {
    If FLASH memory is already configured
    – return without re-initializing
    to avoid destroying existing code
    Else {
        configure Memory Bank control register
        THEN configure Global control
        (turns on the FLASH)
    }
}
```

Look in EZ-Kit documentation

- Don't start from scratch – Look for recommended settings
- These settings are specific for FLASH memory used on the EZ-Kit Lite

Table 2-4. Asynchronous Memory Control Registers Settings Example

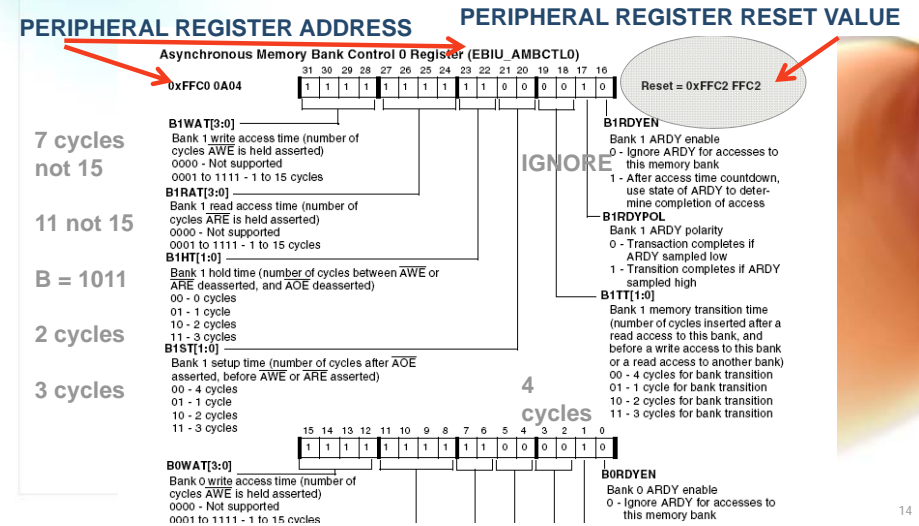
Register	Value	Function
EBIU_AMBCTL0	0x78B078B0	Timing control for Banks 1 and 0
EBIU_AMBCTL1 bits 15-0	0x78B0	Timing control for Bank 2 (Bank 3 is not used)
EBIU_AMGCTL bits 3-0	0xF	Enable all banks Turns on clock

Each Flash chip is initially configured with the memory sectors mapped into the processor's address space as shown in Table 2-5.

13

Set the Bank control register

- Kit documentation recommends 0x78B0. What does this setting do?



14

Control access speed -- REVIEW

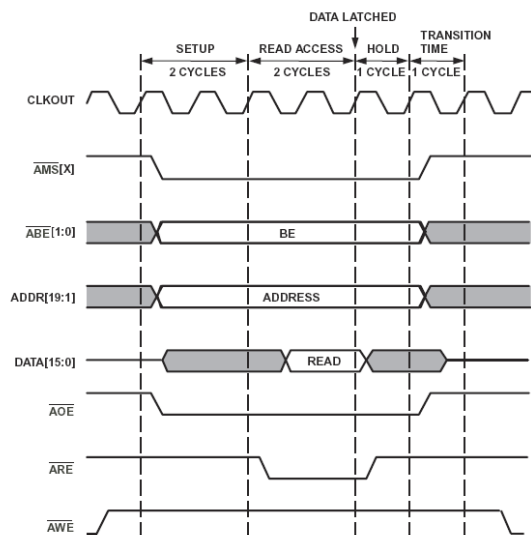
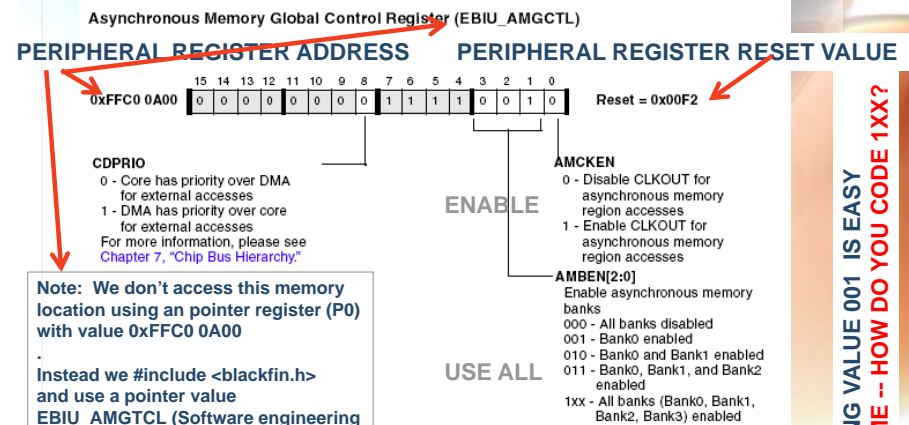


Figure 17-6. Asynchronous Read Bus Cycles

15

Set General Control Register

- Documentation says set to "0xF" for this particular FLASH chip



Note: We don't access this memory location using a pointer register (P0) with value 0xFFC0 0A00
 Instead we #include <blackfin.h> and use a pointer value EBIU_AMGCTL (Software engineering abstraction concept)

Figure 17-3. Asynchronous Memory Global Control Register

CODING VALUE 001 IS EASY ASK ME -- HOW DO YOU CODE 1XX?

16

Key issues -- REVIEW

InitFlashCPP()

Register	Value	Function
EBIU_AMBCTLO	0x7BB07BB0	Timing control for Banks 1 and 0
EBIU_AMBCTL1 bits 15-0	0x7BB0	Timing control for Bank 2 (Bank 3 is not used)
EBIU_AMGCTL bits 3-0	0xF	Enable all banks

Does not sound too big a deal (IN PRINCIPLE)

1. Set pointer to EBIU_AMBCTLO address
2. Then set value = 0x7BB07BB0
3. Then store value at EBIU_AMBCTLO
**pt = value*
4. Then make sure "write occurs" NOW as this processor can delay doing writes "until convenient". This processor is designed to do "writes" when "it is not busy" doing MANY reads

- Do the same for the other "32 bit" FLASH registers

Blackfin 533 I/O

CHANGE INTERFACING FROM ASM TO CPP CODE

Build and Test Stub -- REVIEW

```
// What we want to do -- pseudo-code
// void InitFlashASM(void) {
//     If FLASH memory already configured
//         return without initializing
//     Else {
//         // Order is important
//         configure Memory Bank control register
//         THEN configure Global control
//             (turns on the FLASH)
//     }
// }

// void InitFlashASM(void) {
//     .section program;
//     global _InitFlashASM;
//     InitFlashASM:
//         If FLASH memory already configured
//             return without initializing
//         Else {
//             // Order is important
//             configure Memory Bank control register
//             THEN configure Global control
//                 (turns on the FLASH)
//         }
//     _InitFlashASM.END: RTS;
// }
```

CHANGED TO InitFlashCPP()

Blackfin 533 I/O

CHANGE INTERFACING FROM ASM TO CPP CODE

When stub is tested -- add code

The System Synchronize (SSYNC) instruction forces all speculative, transient states in the core and system to complete before processing continues. Until SSYNC completes, no further instructions can be issued to the pipeline.

The SSYNC instruction performs the same function as Core Synchronize (CSYNC). In addition, SSYNC flushes any write buffers (between the L1 memory and the system interface) and generates a Synch request signal to the external system. The operation requires an acknowledgement Synch_Ack signal by the system before completing the instruction.

```
section program;
global _InitFlashASM;
InitFlashASM:
    If FLASH memory already configured
        return without initializing
    Else {
        // Order is important
        configure Memory Bank control register
    }
// Set P0 to point to EBIU_AMBCTLO
P0 = BIU_AMBCTLO;
// Then R0 = 0x7BB07BB0
R0 = 0x7BB07BB0;
// Then [P0] = R0
[P0] = R0;
// Then make sure write occurs NOW
// as this processor can delay doing writes "until convenient"
SSYNC; // SSYNC Programming manual page 16.8
```

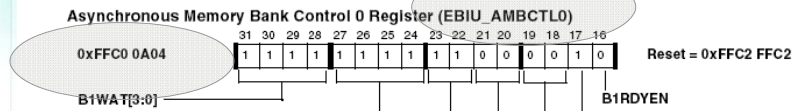
ERROR WHEN WRITING ASM CODE
AVOID SAME PROBLEM IN C++

NEW VIP BLACKFIN
INSTRUCTION SSYNC;
Finish all pipelined operations
before continuing

19

Obvious problem - value needed EBIU_AMBCTLO = ?

- LOOK IN THE MANUAL - CHAPTER 17



```
section program;
global _InitFlashASM;
InitFlashASM:
    If FLASH memory already configured
        return without initializing
    Else {
        // Order is important
        configure Memory Bank control register
    }
// Set P0 to point to EBIU_AMBCTLO
#define EBIU_AMBCTLO = 0xFFC0 0A04
P0 = BIU_AMBCTLO;
// Then R0 = 0x7BB07BB0
R0 = 0x7BB07BB0;
// Then [P0] = R0
[P0] = R0;
// Then make sure write occurs NOW
// as this processor can delay doing writes "until convenient"
SSYNC; // SSYNC Programming manual page 16.8
```

Is the following the correct code to use in C++
#define pBIU_AMBCTLO = (int *) 0xFFC0 0A04

Code review
2 obvious errors
FIND THEM

CHANGE INTERFACING FROM ASM TO CPP CODE

Corrected code – still fails (Get equivalent errors in C++)

The screenshot shows an IDE window with assembly code for `FlashUtilitiesASM.asm`. Annotations with arrows point to specific lines of code:

- `#define EBUI_AMBCTL0 0xFFC00A04`: "No '=' in a define statement"
- `P0 = EBUI_AMBCTL0;`: "No 'spaces' in number"
- `R0 = 0x7BB07BB0;`: "Spell check"
- `[P0] = R0;`: "'0' not 'O'"

Below the code, the error messages are visible:

```

[Error ea5003] ".\FlashUtilitiesASM.asm":25 Semantic Error in instruction :
P0 = 0xFFC00A04;
Operands don't fit instruction template 'REG ASSIGN expr'.
Check for an out of range immediate value or an illegal register.

[Error ea5003] ".\FlashUtilitiesASM.asm":28 Semantic Error in instruction :
R0 = 0x7BB07BB0;
Operands don't fit instruction template 'REG ASSIGN expr'.
Check for an out of range immediate value or an illegal register.
    
```

Additional notes on the right side of the slide:

- DEFECTS in code process**
- Pair programming cost if not caught by partner
- 3 * \$5

MIPS and Blackfin behave same when putting 32 bit numbers into 32 bit registers

- You can't load a 32-bit register with a 32-bit immediate value using one instruction
- **WRONG** `R0 = 0x7BB07BB0;`
- Must load low 16-bit of register
`R0.L = 0x7BB0;`
- Then load high 16-bits of register
`R0.H = 0x7B00;`
- You must load addresses into P0 the same way. You write the code to replace `P0 = 0xFFC00A04`

In C++ code, a similar error is using *unsigned int value (32 bits)* when you meant to use *unsigned short value (16 bits)*

More readable ASM code example

The screenshot shows the same IDE window with a more readable version of the assembly code. Annotations with arrows point to specific lines:

- `#define EBUI_AMBCTL0 0xFFC00A04`: "YOU FIX THIS CODE"
- `R0.L = lo(FLASH_CONTROL0_RESET_VALUE);`
`R0.H = hi(FLASH_CONTROL0_RESET_VALUE);`: "Self documenting code"

Below the code, the error messages are visible:

```

[Error ea5003] ".\FlashUtilitiesASM.asm":26 Semantic Error in instruction :
P0 = 0xFFC00A04;
Operands don't fit instruction template 'REG ASSIGN expr'.
Check for an out of range immediate value or an illegal register.
    
```

Additional notes on the right side of the slide:

- I do "define" and then use (double-click) cut-and-paste the label (AMA -- double-click)

What to look for in the following slides

Detailed look at the *WriteLED()* and *ReadLED()* code you will USE (rather than write) during the familiarization laboratory and assignment 1

- Look at how the Blackfin assembly language syntax is used.
- **KEY ELEMENT TO USE IN LABS AND QUIZZES.**
 - Must do our coding without destroying the operation of existing code functionality.
 - When using hardware, this normally means the extensive use of bitwise AND and OR operations.

WriteFlashLEDASM(long inValue)

USER CASE STUDY – TASK -- Write ‘1’ (on) or ‘0’ (off) to the Port to activate LEDs connected to pins PB5 → PB0, leaving other pins unchanged.

Table 2-9. Flash A Port B Controls

Bit #	User IO	Bit Value
5	LED9	0= LED OFF; 1= LED ON
4	LED8	0= LED OFF; 1= LED ON
3	LED7	0= LED OFF; 1= LED ON
2	LED6	0= LED OFF; 1= LED ON
1	LED5	0= LED OFF; 1= LED ON
0	LED4	0= LED OFF; 1= LED ON

PROGRAMMED IN ASSEMBLY CODE

WriteFlashLEDASM(long in_Value)

1. Read “8-bit LED data register” into 32-bit processor data register R1 (makes a copy)
 2. Keep “top” 2 bits (AND operation on bits 7 and 6) of the copied value in R1
 3. Keep “bottom” 6 bits of “in-par” 32-bit in_value (R0)
 4. OR the two processor data registers
 5. Write “modified copy” back into “LED data register”
- **PROBLEM** “byte” read and writes – how do we do those?

Table 2-6. Flash A Configuration Registers for port A, B

Register Name	Port A Address	Port B Address
Data In (Read-only)	0x2027 0000	0x2027 0001
Data Out (Read-Write)	0x2027 0004	0x2027 0005
Direction (Read-Write)	0x2027 0006	0x2027 0007

26

Standard ENCM369 assembly code problem, but using different syntax

- Start with the stub and pseudo-code of user-case study
 - Use the ‘real C++’ as psuedo-code when we know what to. Use a description otherwise

```
.section program:
.global _WriteFlashLEDASM; // void WriteFlashLEDASM(long in_value);
// WriteFlashLEDASM; // void WriteFlashLEDASM(long in_value);
// PROBLEM "byte" read and writes // unsigned long ledDataCopy;

// Read "LED data register" into processor data register (makes a copy)
// Convert "byte" into "unsigned long" so we can do the math
// Keep "top" 2 bits (AND operation) of copy
// Keep "bottom" 6 bits of "in-par" 32-bit in_value
// OR the two processor data registers
// Write "modified copy" back into "LED data register"

_WriteFlashLEDASM.END: RTS;
```

Type bottom6bitmask

27

Now identify the registers to use

- Input value (In_par) come in R0
- We can use R1, R2 and R3 without saving

```
.section program:
.global _WriteFlashLEDASM; // void WriteFlashLEDASM(long in_value);
// WriteFlashLEDASM; // void WriteFlashLEDASM(long in_value);
// PROBLEM "byte" read and writes // unsigned long ledDataCopy;

// Read "LED data register" into processor data register (makes a copy)
// Convert "byte" into "unsigned long" so we can do the math
// Keep "top" 2 bits (AND operation) of copy
// Keep "bottom" 6 bits of "in-par" 32-bit in_value
// OR the two processor data registers
// Write "modified copy" back into "LED data register"

_WriteFlashLEDASM.END: RTS;
```

Type bottom6bitmask

28

Add in the code we understand

```

.section program;
.global _WriteFlashLEDASM;          // void WriteFlashLEDASM(long in_value);
//                                     ^^ in R0
#define in_value_R0 R0

_WriteFlashLEDASM:
// PROBLEM "byte" read and writes
// unsigned long ledDataCopy;
#define ledDataCopy_R1 R1
// Read "LED data register" into processor data register (makes a copy)
// Convert "byte" into "unsigned long" so we can do the math
// ??????

// Keep "top" 2 bits (AND operation) of copy
// #define TOP2BITS_MASKVALUE 0xC
#define TOP2BITS_MASKVALUE 0xC      // unsigned long top2BitMask = TOP2BITS_MASKVALUE;
#define top2BitMask_R2 R2
top2BitMask_R2 = TOP2BITS_MASKVALUE;

ledDataCopy_R1 = LedDataCopy_R1 & top2BitMask_R2;
// R2 is now dead -- could re-use

// Keep "bottom" 6 bits of "in-par" 32-bit in_value
// #define BOTTOM6BITS_MASKVALUE 0x3F
#define BOTTOM6BITS_MASKVALUE 0x3F  // unsigned long bottom6BitMask = BOTTOM6BITS_MASKVALUE;
#define bottom6BitMask_R3 R3
bottom6BitMask_R3 = BOTTOM6BITS_MASKVALUE;

in_value_R0 = in_value_R0 & bottom6BitMask_R3;
// R3 is now dead -- could reuse

// OR the two processor data registers
// ledDataCopy = ledDataCopy | in_value;
ledDataCopy_R1 = ledDataCopy_R1 | in_value_R0;

// Write "modified copy" back into "LED data register"
// ?????????

_WriteFlashLEDASM.END: RTS;
    
```

Fixed typo

Still another syntax problem

8 bit and 32 bit writes

(Chapter 6 of instruction user manual?)

- **[P0] = R0;** 32-bit write (4 bytes)
 - Places all 32-bits of processor data register into “long word” (32 bit) address starting at memory location P0
 - If P0 = 0x1000 – then place “32-bit” value at memory location 0x1000
- **B[P0] = R0;** 8-bit write
 - Places “bottom” 8-bits of 32-bit processor data register into “byte” (8 bit) address starting at memory location pointed to by pointer register P0

COMMON MIS-UNDERSTANDING -- You need a 32-bit (not 8-bit) address register to write a 8 bit data value (because memory is large and so you need a large address to access)

Blackfin BF533 I/O

8 bit and 32 bit reads

- **R0 = [P0];** 32-bit read (4 bytes)
 - Places all 32-bits of “long word” (32 bit) address starting at memory location P0 into processor data register
 - If P0 = 0x1000 – then place “32-bit” value at memory location 0x1000
- **R0 = B[P0] (Z);** 8-bit read
 - Places “byte” (8 bit) address starting at memory location P0 into “bottom” 8-bits of processor data register and puts “0” into the “top” 24 bits of register
 - Must convert “8-bit” read operation into a “32” bit “store in register” operation

COMMON MIS-UNDERSTANDING -- You need a 32-bit (not 8-bit) address register read a 8 bit data value (because memory is large and so you need a large address to access)

Add byte read and write operations

```

.section program;
.global _WriteFlashLEDASM;          // void WriteFlashLEDASM(long in_value);
//                                     ^^ in R0
#define in_value_R0 R0

_WriteFlashLEDASM:
// PROBLEM "byte" read and writes
// unsigned long ledDataCopy;
#define ledDataCopy_R1 R1
// Read "LED data register" into processor data register (makes a copy)
// Convert "byte" into "unsigned long" so we can do the math
#define LED_DATA_REGISTER_ADDRESS 0x2027_0008
P0_L = lo(LED_DATA_REGISTER_ADDRESS); P0_H = hi(LED_DATA_REGISTER_ADDRESS)
ledDataCopy_R1 = B[P0] (Z);

// Keep "top" 2 bits (AND operation) of copy
// #define TOP2BITS_MASKVALUE 0xC
#define TOP2BITS_MASKVALUE 0xC      // unsigned long top2BitMask = TOP2BITS_M
#define top2BitMask_R2 R2
top2BitMask_R2 = TOP2BITS_MASKVALUE;

ledDataCopy_R1 = LedDataCopy_R1 & top2BitMask_R2;
// R2 is now dead -- could re-use

// Keep "bottom" 6 bits of "in-par" 32-bit in_value
// #define BOTTOM6BITS_MASKVALUE 0x3F
#define BOTTOM6BITS_MASKVALUE 0x3F  // unsigned long bottom6BitMask = BOTTOM6
#define bottom6BitMask_R3 R3
bottom6BitMask_R3 = BOTTOM6BITS_MASKVALUE;

in_value_R0 = in_value_R0 & bottom6BitMask_R3;
// R3 is now dead -- could reuse

// OR the two processor data registers
// ledDataCopy = ledDataCopy | in_value;
ledDataCopy_R1 = ledDataCopy_R1 | in_value_R0;

// Write "modified copy" back into "LED data register"
B[P0] = ledDataCopy_R1;

_WriteFlashLEDASM.END: RTS;
    
```

Is this correct for keeping top 2 bits of an 8-bit value?

“DEFECT” if not corrected
** AMW **

Still syntax problems
“ERRORS”

Fix as exercise

Test by using instead of my code in Labs

InitFlashPortCPP() to complete

- Set direction to 1 on lower pins leaving other direction values unchanged
 - Read “direction” byte register into processor data register (makes a copy)
 - Set another processor data register to 0x3F
 - OR the two data registers (HOW?)
 - Write “modified copy” back into “direction byte register”

Table 2-6. Flash A Configuration Registers for port A, B

Register Name	Port A Address	Port B Address
Data In (Read-only)	0x2027 0000	0x2027 0001
Data Out (Read-Write)	0x2027 0004	0x2027 0005
Direction (Read-Write)	0x2027 0006	0x2027 0007

NOW USE CPP VERSION OF THIS CODE

Agenda

- Processors need to send out control signals (high / low 1 / 0 true / false)
 - General purpose input / output GPIO on processor chip (16)
 - FLASH memory chip has additional I/O ports connected to Ez-Lite KIT LED's
- Making the FLASH memory I/O port control the Ez-KIT LED's
- The new Blackfin assembly language instructions needed