Record Member alignment PIC24/dsPIC and PIC32

2021-06-13

1. Contents

2. Introduction

A while ago I ran into a problem related to variable alignment. I tried to read in the data of a record transmitted via Uart by a PIC18 into a PIC32. It went wrong because the internal structure of the record was different between the two PIC families.

Apparently also inside a variable alignment is taken into account.

3. Excerpts from the mP compiler's help:

3.1 PIC24/dsPIC: Variable, constant and routine alignment

Simple type variables whose size exceeds 1 byte (`word`, `integer`, `dword`, `longint`, `real`) are always set to alignment 2 (i.e. are always allocated on even address).
Derived types and constant aggregates whose at least one element exceeds size of 1 byte are set to alignment 2.
Routines are always set to alignment 2.

3.2 Pic32: Variable, constant and routine alignment

Simple type variables whose size is 2 bytes are set to alignment 2, those whose size is 4 bytes and larger are set to alignment 4. Variables of other size are set to alignment 1.
Routines are always set to alignment 4. Only aligned memory access is supported.

4. Example

4.1 Code

```
program Word_Alignment;
{ Declarations section }
Type TArr = Array[5] of byte;
Type Trec = record
    Byte1 : byte;
    Word1 : word;
    Byte2 : byte;
    Dword1: dword;
    Byte3 : byte;
    Arr1  : TArr;
    end;

var MyRec: TRec;  volatile;
    Siz: byte;

begin
  { Main program }
  Siz := SizeOf(MyRec);
end.
```

4.2 Result

Address	Pic18	PIC24/dsPIC	PIC32	
$+0	Byte1	Byte1	Byte1	← alignment 2 and 4
+1	Word1			
+2		Word1	Word1	← alignment 2
+3	Byte2			
+4	Dword1	Byte2	Byte2	← alignment 2 and 4
+5				
+6		Dword1		← alignment 2
+7				
+8	Byte3		Dword1	← alignment 2 and 4
+9	Arr1			
+10		Byte3		← alignment 2
+11		Arr1		
+12			Byte3	← alignment 2 and 4
+13			Arr1	
+14				← alignment 2
+15				
+16				← alignment 2 and 4
+17				

'$' is the baseaddress of the record variable. Bytes marked gray are inserted to achieve the necessary alignment for the following member.

Alignment 2 means 'address mod 2 = 0', aligmnent 4 means 'address mod 4 = 0'.

As one can see, on PIC18 the record variable takes 14 bytes (no variable alignment at all), on PIC24/dsPIC it takes 16 bytes and on PIC32 it takes 18 bytes. This can give problems when e.g. writing the record to MMC/SD card (e.g. 14 bytes from P18) and reading it back into a PIC32 (which expects 18 bytes).

Furthermore the position in the record (offset) differs a lot for all record members except the first one (Byte1). This is because the record members have to be aligned due to the rules in sections 3.1 and 3.2.

This means also that problems can arise when accessing record members in other ways than via their member name: one can not be sure where members reside inside the record, e.g. via pointers to another type of variable (e.g. an array of bytes).

In principle the compiler is free to place a record member anywhere in the record. Until now the order in the record a made the same as the order of the members in the recordtype definition.

5. Another example (structure “TbootInfo” from the mE Bootloader)

5.1 Code

```
(* Bootloader info field types *)

// Byte field (1 byte).
type TCharField = record
  fFieldType: byte;
  fValue    : byte;
end;

// Int field (2 bytes).
type TUIntField = record
  fFieldType: byte;
  intVal    : word;
end;

// Long field (4 bytes).
type TULongField = record
  fFieldType: byte;
  fValue    : dword;
end;

// String field (MAX_STRING_FIELD_LENGTH bytes).
const MAX_STRING_FIELD_LENGTH = 20;
type TStringField = record
  fFieldType: byte;
  fValue    : string[MAX_STRING_FIELD_LENGTH];
end;

// Bootloader info record (device specific information).
type TBootInfo = record
  bSize       : byte;          (1)
  bMcuType    : TCharField;    (2)
  ulMcuSize   : TULongField;   (3)
  uiEraseBlock: TUIntField;    (4)
  uiWriteBlock: TUIntField;    (5)
  uiBootRev   : TUIntField;    (6)
  ulBootStart : TULongField;   (7)
  sDevDsc     : TStringField;  (8)

end;
```

As one can see, each member of TbootloaderInfo is itself a structure and starts with a byte followed by either a byte, a word, a dword or a string.

The structure “Tbootloaderinfo” is casted to an array of byte and sent from the PIC bootloader to the PC based tool “USB HID Bootloader”. This is where extra bytes are inserted for the P24 and P32 bootloaders to achieve the necessary alignment.

5.2 Result

Address $	PIC18	PIC24/dsPIC	PIC32	
$+0	(1)	(1)	(1)	← alignment 2 and 4
+1	(2)	(2)	(2)	
+2				← alignment 2
+3	(3)			
+4		(3)	(3)	← alignment 2 and 4
+5				
+6				← alignment 2
+7				
+8	(4)			← alignment 2 and 4
+9				
+10		(4)		← alignment 2
+11	(5)			
+12			(4)	← alignment 2 and 4
+13				
+14	(6)	(5)		← alignment 2
+15				
+16			(5)	← alignment 2 and 4
+17	(7)			
+18		(6)		← alignment 2
+19				
+20			(6)	← alignment 2 and 4
+21				
+22	(8)	(7)		← alignment 2
+23				
+24			(7)	← alignment 2 and 4
+25				
+26				← alignment 2
+27				
+28		(8)		← alignment 2 and 4
+29				
+30	…			← alignment 2
+31				
+32			(8)	← alignment 2 and 4
		…		← alignment 2
				← alignment 2 and 4
			…	

Legend: Green = leading byte in each record member, Gray = interted byte to make the next field aligned, Yellow = datafield following the leading byte.

As one can see for the P18 no extra bytes are inserted, for the P24 and the P32 extra bytes are inserted to meet the citeria sections 3.1 and 3.2.

6. Tips

- The safest way to access a record member is via its Member Name.
- In case of serial transmission of the record content (e.g. Uart or mmc/sd Card) make sure that the actual number of bytes of the record variable are taken into account (can be more than the simple addition of the sizes of the members), and also that something is done about the not always clear offset of a member in the record: the serial transmission should take into account the dummy bytes inserted to achieve alignment. Of course you can also send or write all bytes (including the dummies) to uart/card. In the latter case read back to another PIC family will not be possible without some correction.
- If you are obliged to hold e.g. an array of a recordtype in ram then it is important that the recordtype takes as less bytes as possible. You can make this happen by re-ordering the members of your record: first the Dwords (alignment 4), then the Words (alignment 2) and then the rest. Acting this way the same record size of that from a PIC18 can be realized.

7. Links

- Forum posts with problem reports:

https://forum.mikroe.com/viewtopic.php?f=106&t=72181&p=305458#p305458

https://forum.mikroe.com/viewtopic.php?f=172&t=61287

https://forum.mikroe.com/viewtopic.php?f=106&t=72125

- A tool showing the alignment of variables (Thanks VCC)

https://libstock.mikroe.com/projects/view/2434/misaligned-address-finder

[end of ducument]