HCS12X – Data Definition

The present document describes how programmer can help the HCS12X compiler to generate the more optimal code for data access. It will cover following topics:

- Variables allocated in direct addressing area
- Variables allocated in extended addressing area
- Variables allocated in banked addressing area – Using Logical Addresses
- Variables allocated in banked addressing area – Using Global Addresses
- Banked Constant allocation
- Logical Addresses vs. Global Addresses

For each of the variable type enumerated above, we will describe:

- How to define a variable
- How to declare a variable
- Code generated to access the variable
- How to define a pointer pointing to such a variable
- Code generated to access the pointer and access the variable.
- Placement in PRM file.

Note 1:
Information described in this technical note apply to SMALL (-Ms) and BANKED (-Mb) Memory model. They do not apply to LARGE (-Ml) memory model.

Note 2:
We usually recommend using SMALL memory model for application with less than 32Kb code and BANKED memory model otherwise. We do not recommend using LARGE memory model.

Variables allocated in Direct Addressing Area

In order to inform the compiler that a variables is allocated on the direct page, you have to define (and declare) it in a specific segment with attribute __SHORT_SEG.

- Defining & Accessing Data on Direct Addressing Area:

1. Variable definition is done as follows:
```
#pragma DATA_SEG __SHORT_SEG MyShortData
unsigned char  rub_short_var;
#pragma DATA_SEG DEFAULT
```
2. Variable declaration is done as follows:
```
#pragma DATA_SEG __SHORT_SEG MyShortData
extern unsigned char  rub_short_var;
#pragma DATA_SEG DEFAULT
```
3. Access to the variable will generate following code:
```
    37: rub_short_var = 2;
00E08002 C602          [1] LDAB  #2
00E08004 5B08          [2] STAB  $08
```
4. There is no special pointer type for variables allocated in the direct page[1]. A pointer pointing to such a variable will be defined as follows:
```
unsigned char* ptr_on_short_var;
```
5. Initializing and accessing the pointed object will then generate the code below:
```
    46: ptr_on_short_var = &rub_short_var;
00E0801F 180320102102 MOVW  #8208,$2102
    47: (*ptr_on_short_var)++;
00E08025 62FBA0D9     INC   [$A0D9,PC] /* [ptr_on_short_var,PCR] */
```
6. For variable defined in a __SHORT_SEG section, SEGMENT and PLACEMENT will be done as follows in the PRM file
```
SEGMENTS
  DIRECT_PAGE     = READ_WRITE      0x2010 TO   0x20FF;
  /* Other segment definition here*/
END
PLACEMENT
  MyShortData           INTO DIRECT_PAGE;
  /* Other placement definition here*/
END
```

Note:

Section containing variables accessed using direct addressing mode should be allocated in segment defined with logical addresses.

Configuring Direct Addressing Area:

On HCS12X, the direct page can be moved and is not hard-coded to 0x00..0xFF (as it was on HCS12). The compiler does require a special setup to support this:

- The compiler and assembler option -CpDirect must be used with the starting address of the Direct accessible area if the DIRECT register contains anything but the default value 0.
  E.g. If DIRECT is initialized with 0x20, then the Direct window is from 0x2000 up to 0x20FF and

1 This assumes the usual SMALL or BANKED memory model. In the usually not necessary LARGE memory model, pointers must be qualified __near.

the compiler and assembler option is -CpDirect0x2000. Add this option to both the compiler and to the assembler settings.

- The DIRECT register must be initialized by the user code. The default startup code does not initialize DIRECT.
- In the prm file, the section MyShortData has to be allocated accordingly to this area(in the example from 0x2000 to 0x20FF).
- Be careful, there is no linker diagnostic message for an incorrect allocation of the direct section (say if MyShortData is not allocated correctly).
  For the HCS12, the linker issues a fixup overflow. But for the HCS12X this is no longer possible as the direct page can be mapped.

Variables allocated in Extended Addressing Area

In order to inform the compiler that a variables is allocated on the extended address space, you just need to define (and declare) it the usual way[2].

1. Variable definition is done as follows:
```
unsigned char  rub_var;
```
2. Variable declaration is done as follows:
```
extern unsigned char rub_var;
```
3. Access to the variable will generate following code:
```
    39: rub_var =7;
00E08002 C607          [1] LDAB  #7
00E08004 7B2001        [3] STAB  $2001
```
4. A pointer pointing to such a variable can be defined as follows:
```
unsigned char  * ptr_on_var;
```
5. Initializing and accessing the pointed object will then generate the code below:
```
    57: ptr_on_var = &rub_var;
00E0805F 18032100210C MOVW  #8448,$210C
    58: (*ptr_on_var)++;
00E08065 62FBA0A3     INC   [$A0A3,PC] /* [ptr_on_var,PCR] */
```
6. Variable defined this way are allocated in predefined section DEFAULT_RAM. In the PRM file, SEGMENT and PLACEMENT definition for this section will be done as follows:
```
SEGMENTS
   RAM       = READ_WRITE      0x2100 TO   0x3FFF;
/* Other segment definition here*/
END
PLACEMENT
          DEFAULT_RAM       INTO  RAM;
  /* Other placement definition here*/
END
```

Note:

Section containing variables accessed using extended addressing mode should be allocated in segment defined with logical addresses.

Variables allocated in Banked Addressing Area

Variables allocated in banked RAM can be accessed using

- Logical addresses (using RPAGE) or

2 This assumes the usual SMALL or BANKED memory model. In the usually not necessary LARGE memory model, extended variables sections must be qualified with __NEAR_SEG and pointers with __near.

- Global addresses.

Using Logical Addresses

In order to tell the compiler you want to access a variable using his logical address you have to use the following notation:

1. Variable definition is done as follows:
```
#pragma DATA_SEG __RPAGE_SEG PAGED_RAM
unsigned char  rub_far_var;
#pragma DATA_SEG DEFAULT
```
2. Variable declaration is done as follows:
```
#pragma DATA_SEG __RPAGE_SEG PAGED_RAM
extern unsigned char  rub_far_var;
#pragma DATA_SEG DEFAULT
```
3. Access to the variable will generate following code:
```
    38: rub_far_var = 5;
00E08002 C6FB          [1] LDAB  #251 /* #PAGE(rub_far_var) */
00E08004 5B16          [2] STAB  $16  /* RPAGE */
00E08006 C605          [1] LDAB  #5
00E08008 7B1000        [3] STAB  $1000
```
4. A pointer pointing to such a variable can be defined as follows:
```
unsigned char  * __rptr  ptr_on_far_var;
```
5. Initializing and accessing the pointed object will then generate the code below:
```
    50: ptr_on_far_var = &rub_far_var;
00E08029 180310002105 MOVW  #4096,$2105
00E0802F 180BFB2104   MOVB  #251,$2104
    51: (*ptr_on_far_var)++;
00E08034 FE2105       LDX   $2105    /* ptr_on_far_var:1 */
00E08037 F62104       LDAB  $2104    /* ptr_on_far_var */
00E0803A 7B0016       STAB  $0016   /* RPAGE */
00E0803D 6200         INC   0,X
```
6. For variable defined in a RPAGE section, SEGMENT and PLACEMENT will be done as follows in the PRM file:
```
SEGMENTS
  RAM_FB          = READ_WRITE    0xFB1000 TO 0xFB1FFF;
  RAM_FC          = READ_WRITE    0xFC1000 TO 0xFC1FFF;
  RAM_FD          = READ_WRITE    0xFD1000 TO 0xFD1FFF;
/* Other segment definition here*/
END
PLACEMENT
  PAGED_RAM         INTO  RAM_FB, RAM_FC, RAM_FD;
  /* Other placement definition here*/
END
```

Note:

Section containing variables accessed using logical addressing mode should be allocated in segment defined with logical addresses.

Note:

Variables allocated in a RPAGE segment can alternatively be accessed using far pointers.

```
unsigned char  * __far  ptr_on_far_var;
```

In this case global addressing mode will be used to access the pointed object.

Using Global Addresses

The main reason to use global addresses is because an object may not fit into a single page of a logical address. With objects accessed with Global Addresses, this limitation can be avoided and objects up to the available memory size or up to 64kB can be accessed.
Using Global Addresses for pointers (== __far pointer) can be used for any object, objects do not need to be qualified in any particular way.
In order to tell the compiler you want to access a variable using his global address you have to use the following notation:

1. Variable definition is done as follows:
```
#pragma DATA_SEG __GPAGE_SEG PAGED_RAM
unsigned char  rub_far_var;
#pragma DATA_SEG DEFAULT
```
2. Variable declaration is done as follows:
```
#pragma DATA_SEG __GPAGE_SEG PAGED_RAM
extern unsigned char  rub_far_var;
#pragma DATA_SEG DEFAULT
```
3. Access to the variable will generate following code:
```
    35: rub_far_var = 7;
00E0802B C607          [1] LDAB  #7
00E0802D 860F          [1] LDAA  #15 /* #GLOBAL_PAGE(rub_far_var)*/
00E0802F 5A10          [2] STAA  $10 /* GPAGE*/
00E08031 187BE00F      [4] GSTAB $E00F
```
4. A pointer pointing to such a variable will be defined as follows[3]:
```
unsigned char  *__far  ptr_on_far_var;
```
5. Initializing and accessing the pointed object will then generate the code below:
```
    37: ptr_on_far_var = &rub_far_var;
00E08035 1803B0002101 MOVW  #45056,$2101
00E0803B 180B0F2100   MOVB  #15,$2100
    38: (*ptr_on_far_var)++;
00E08040 FE2101       LDX   $2101  /* ptr_on_far_var:1 */
00E08043 F62100       LDAB  $2100  /* ptr_on_far_var */
00E08046 5B10         STAB  $10    /* GPAGE */
00E08048 18A600       GLDAA 0,X
00E0804B 42           INCA
00E0804C 186A00       GSTAA 0,X
```
6. For variable defined in a GPAGE section, SEGMENT and PLACEMENT will be done as follows in the PRM file:
```
SEGMENTS
  RAM_BANKED      = NO_INIT     0xF9000'G TO 0xFCFFF'G;
  /* Other segment definition here*/
END
PLACEMENT
  PAGED_RAM         INTO RAM RAM_BANKED;
  /* Other placement definition here*/
END
```

Note:

Section containing variables accessed using global addressing mode can be allocated in segment defined with either logical or global addresses. So alternatively PRM file can look as follows:
```
SEGMENTS
```

3 Actually __far pointer can point to any object, not just to objects allocated in a __GPAGE_SEG qualifier segment.

```
  RAM_FB          = READ_WRITE    0xFB1000 TO 0xFB1FFF;
  RAM_FC          = READ_WRITE    0xFC1000 TO 0xFC1FFF;
  RAM_FD          = READ_WRITE    0xFD1000 TO 0xFD1FFF;
/* Other segment definition here*/
END
PLACEMENT
  PAGED_RAM         INTO RAM_FB, RAM_FC, RAM_FD;
  /* Other placement definition here*/
END
```

Paged Constant allocation

Constants can be allocated in banked EEPROM or in banked FLAH and can be accessed using

- Logical addresses in EEPROM(using EPAGE)or
- Logical addresses in FLASH (using PPAGE) or
- Global addresses

Using Logical Addresses in EEPROM

In order to tell the compiler you want to access a constant using his logical address in EEPROM you have to use the following notation:

1. Constant definition is done as follows:
```
#pragma CONST_SEG __EPAGE_SEG PAGED_CONST
const unsigned char cub_far_const=1;
#pragma CONST_SEG DEFAULT
```
2. Constant declaration is done as follows:
```
#pragma CONST_SEG __EPAGE_SEG PAGED_CONST
extern const unsigned char cub_far_const=1;
#pragma CONST_SEG DEFAULT
```
3. A pointer pointing to such a variable will be defined as follows:
```
const unsigned char *__eptr ptr_on_far_const;
```

Note:

Constants allocated in a EPAGE segment can alternatively be accessed using far pointers.
`const unsigned char *__far ptr_on_far_const;`
In this case global addressing mode will be used to access the pointed object.

Using Logical Addresses in FLASH

Using Logical Addresses in FLASH is supported by the compiler only for code which is not allocated in any paged area. As this is often not the case, we recommend to use Global addressing for all data objects in FLASH.
To implement using logical addresses, use the segment qualifier __PPAGE_SEG and the pointer qualifier __pptr.

Using Global Addresses

1. Constant definition is done as follows:
```
#pragma CONST_SEG __GPAGE_SEG PAGED_CONST
const unsigned char cub_far_const=1;
#pragma CONST_SEG DEFAULT
```
2. Constant declaration is done as follows:
```
#pragma CONST_SEG __GPAGE_SEG PAGED_CONST
extern const unsigned char cub_far_const= 1;
```

```
#pragma CONST_SEG DEFAULT
```

3. A pointer pointing to such a variable will be defined as follows:

```
const unsigned char *__far ptr_on_far_const;
```

Logical Addresses vs. Global Addresses

- For variables which are not larger than 4K (the size of a Banked RAM window), using logical addresses is more efficient than using global addresses.

	Logical Address	Global Address
Data access (Code Size) xy = 3;	9 Bytes	10 Bytes
Data access (Execution Speed) xy = 3	7 Cycles	8 Cycles
Pointer access (Code Size) *xx = 4	11 Bytes	12 Bytes
Pointer access (Execution Speed) *xx = 4	11 Cycles	12 Cycles
Pointer content increment (Code Size) (*xx)++	11 Bytes	15 Bytes
Pointer content increment (Execution Speed) (*xx)++	12 Cycles	16 Cycles

- If a variable is bigger than 4K, the linker will have to allocate it across a bank boundary. In this case, we recommend to use global addressing to access the variable.
- We recommend using Global Addressing mode to access constants or string constants allocated in FLASH.
- We recommend using logical addresses for all objects, which use logical addresses at runtime. This includes the stack, code, direct variables, extended variables, I/O registers.
- In order to define a paged variable, make sure to use a DATA_SEG pragma. Do not attempt to use __far keyword in this purpose. The far keyword indicates how the compiler should access a variable, it does not change the way variables are allocated.