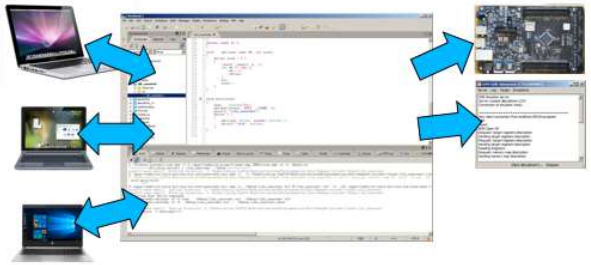
Programbibliotek

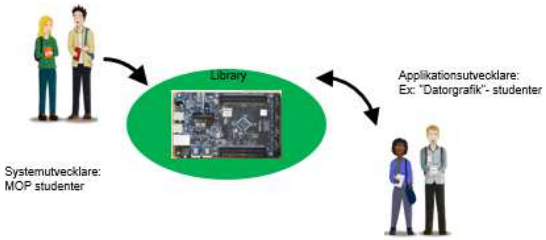


Ur innehållet:

Exekveringsmiljö

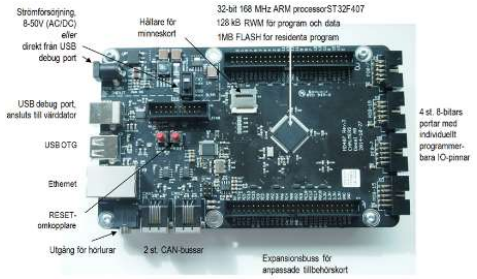
Olika typer av programbibliotek

- Kompilatorbibliotek
- C-bibliotek



Läsanvisningar:

Arbetsbok kapitel 8



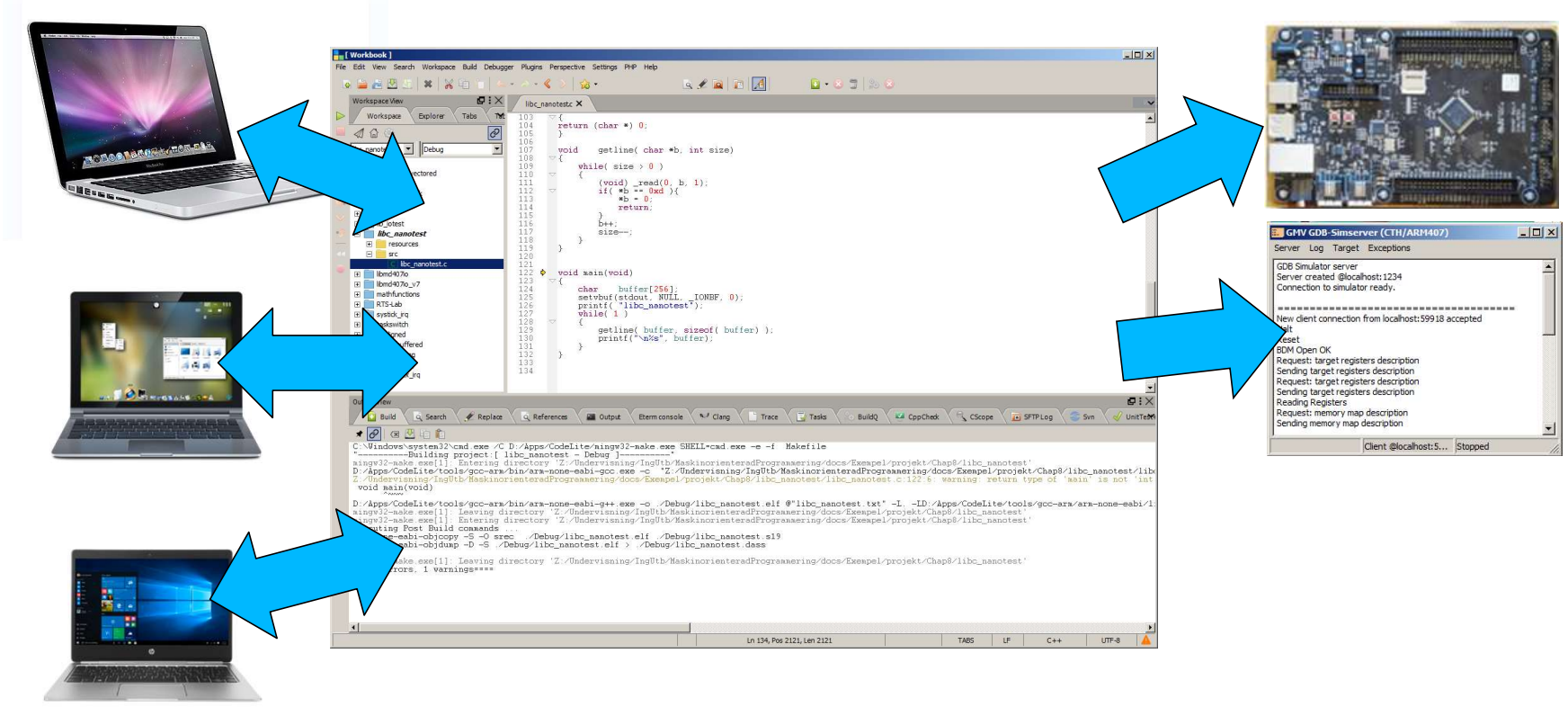
Målsättningar:

Att kunna använda färdiga programbibliotek

Att kunna skapa och underhålla ett nytt programbibliotek

Värddator och måldator:

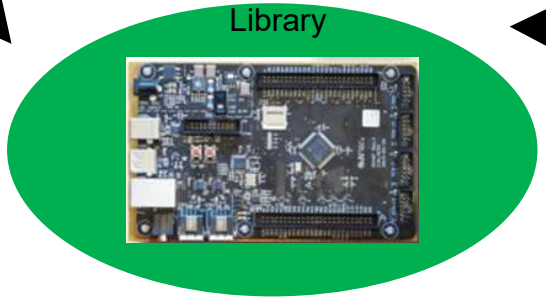
Konventionell programutveckling sker *på värddator för värddator*.
Korsutveckling sker *på värdator för måldator*, i vårt fall MD407.



System-/Applikations- programmering för MD407

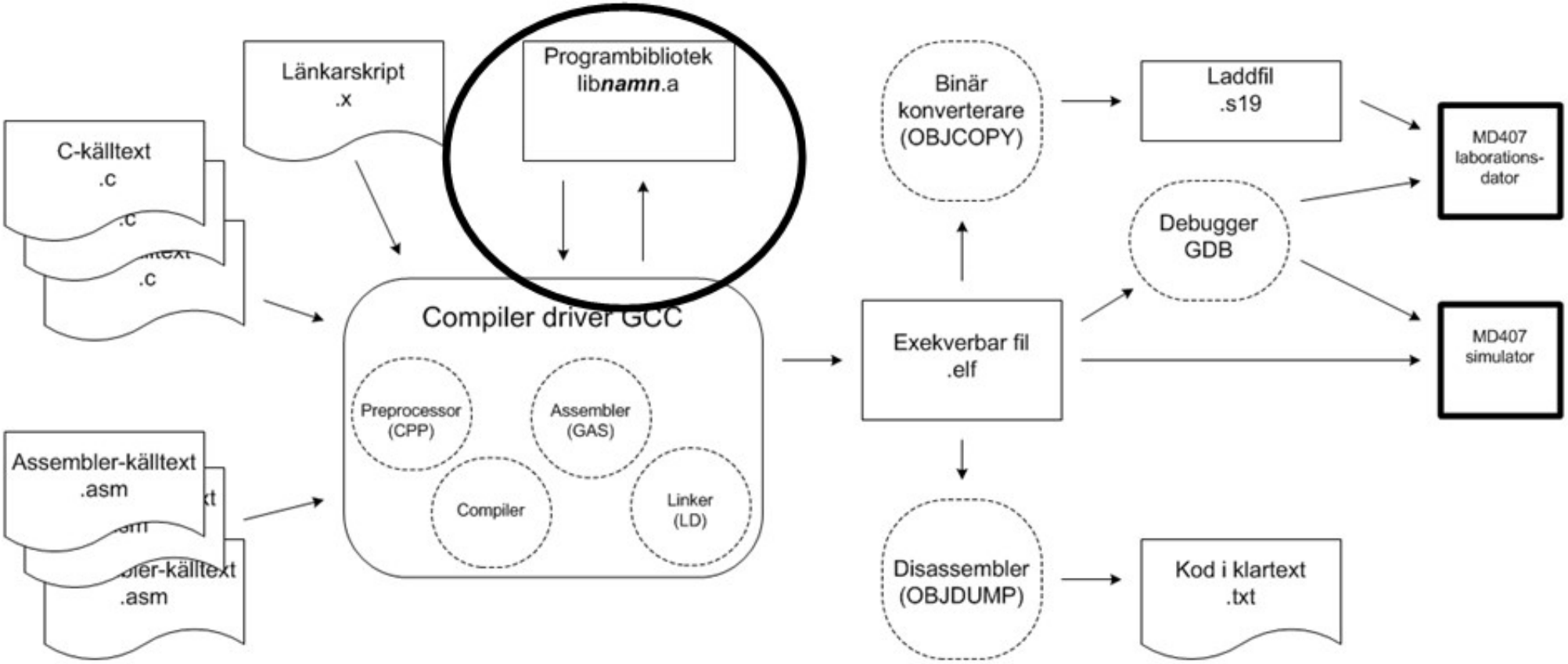
Med *programbibliotek*, avser vi att skapa grundläggande funktionalitet för användningen av vårt datorsystem. *Exempel: implementering av grundläggande operationer i programvara. In-/ut-matning, filhantering, fönsterhantering etc.*
Med *systemprogramvara* menar vi oftast "basal" programvara för vår dator, ett programlager med generellt användbar natur. *Exempel: operativsystem och drivrutiner.*
Med *applikationsprogramvara*, menar vi program avsedda för mer specifik användning.
Exempel: Spel, navigation, övervakning/styrning etc.

Systemutvecklare:
MOP studenter



Applikationsutvecklare:
Ex: "Datorgrafik"- studenter

Programbibliotek



Programbibliotek – olika typer

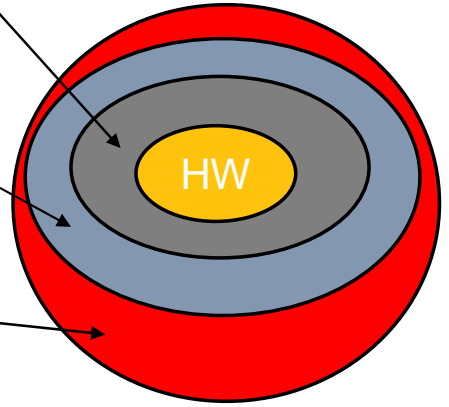
Programbibliotek kan ofta ses som "kompatibilitetslager" där en anpassning sker till någon speciell maskinvara och något specifikt operativsystem.

- Kompilatorbibliotek – tillhandahåller det stöd som krävs för att ett standardiserat C-program korrekt ska kunna översättas till maskinkod, oavsett processorarkitektur.
- Standard C-bibliotek – tillhandahåller en lång rad funktioner användbara i de flesta applikationer. Utgör också vanligtvis det viktigaste gränssnittet mot det använda operativsystemet.
- Användarspecifika bibliotek.

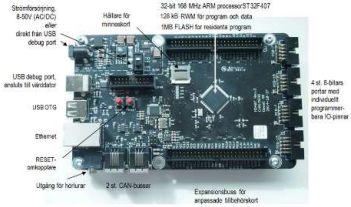
Exempel: Flyttalsoperationer

Exempel: `malloc/free, printf…`

Exempel: GUI: Fönster och menyer



Måldator – arkitektur

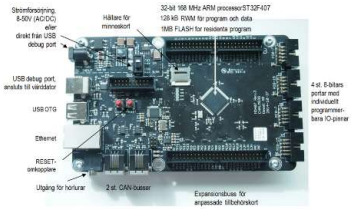


En måldator kan karakteriseras av:
arkitektur, *minnesdisposition* och *periferienheter*.

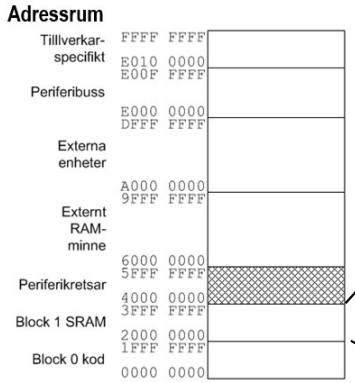
- Arkitektur – ARM-Cortex, tillhandahåller processorer som hanterar olika typer av instruktionsuppsättningar:
  - Thumb 1
  - Thumb 2
  - DSP (*Digital Signal Processing*)
  - FP (*Floating Point*)

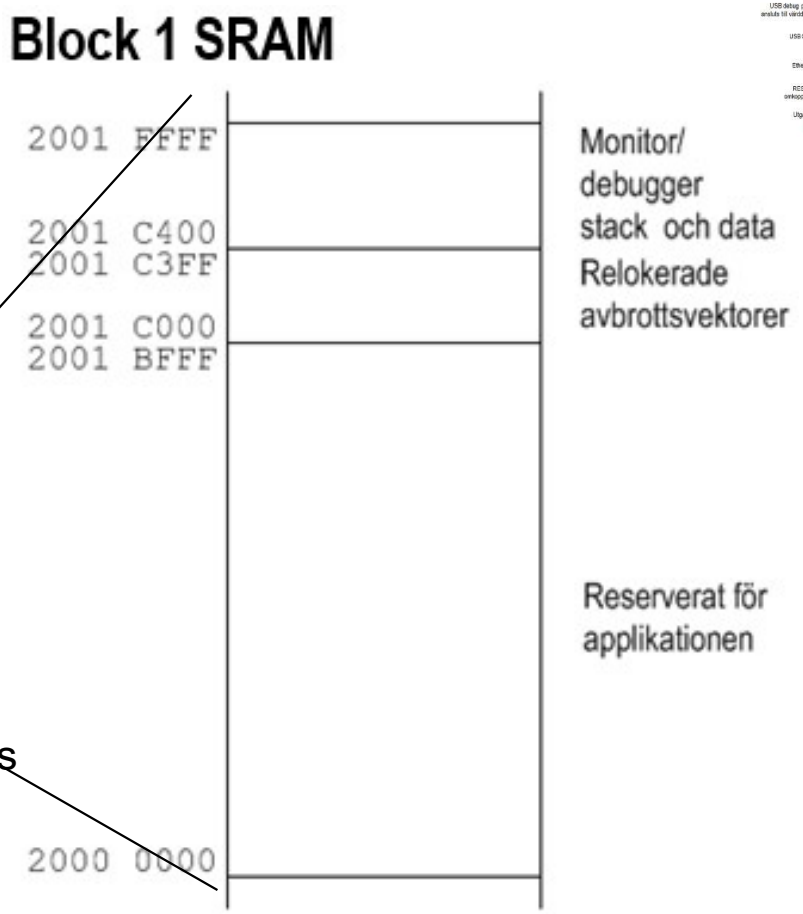
Instruktion	Storlek	Cortex M0	Cortex M0+	Cortex M1	Cortex M3	Cortex M4	Cortex M7	Ark.
ADC, ADD, (ADR), AND, ASR, B, BIC, BKPT, BLX, BX, CMN, CMP, CPS, EOR, LDM, (LDMIA, LDMFD), LDR, LDRB, LDRH, LDRSB, LDRSH, LSL, LSR, MOV, MUL, MVN, NOP, ORR, POP, PUSH, REV, REV16, REVSH, ROR, RSB, SBC, SEV, STM, (STMIA, STMEA), STR, STRB, STRH, SUB, SVC, SXTB, SXTH, TST, UXTB, UXTH, WFE, WFI, YIELD	16-bit	x	x	x	x	x	x	v6
BL, DMB, DSB, ISB, MRS, MSR	32-bit	x	x	x	x	x	x	
CBNZ, CBZ, IT	16-bit				x	x	x	
ADC, ADD, AND, ASR, B, BFC, BFI, BIC, CDP, CLREX, CLZ, CMN, CMP, DBG, EOR, LDC, LDMA, LDMDB, LDR, LDRB, LDRBT, LDRD, LDREX, LDREXB, LDREXH, LDRH, LDRHT, LDRSB, LDRSBT, LDRSHT, LDRT, MCR, LSL, LSR, MLS, MCRR, MLA, MOV, MOVT, MRC, MRRC, MUL, MVN, NOP, ORN, ORR, PLD, PLDW, PLI, POP, PUSH, RBIT, REV, REV16, REVSH, ROR, RRX, RSB, SBC, SBFX, SDIV, SEV, SMLAL, SMULL, SSAT, STC, STMDB, STR, STRB, STRBT, STRD, STREX, STREXB, STREXH, STRH, STRHT, STRT, SUB, SXTB, SXTB, TBB, TBH, TEQ, TST, UBFX, UDIV, UMLAL, UMULL, USAT, UXTB, UXTH, WFE, WFI,YIELD	32-bit				x	x	x	v7
PKH, QADD, QADD16, QADD8, QASX, QDADD, QDSUB, QSAX, QSUB, QSUB16, QSUB8, SADD16, SADD8, SASX, SEL SHADD16, SHADD8, SHASX, SHSAX, SHSUB16, SHSUB8, SMLABB, SMLABT, SMLATB,SMLATT, SMLAD, SMLALBB, SMLALBT, SMLALTB, SMLALTTSMLALD, SMLAWB, SMLAWT, SMLSD, SMLSLD, SMMLA, SMMLS, SMMUL, SMUAD, SMULBB, SMULBT, SMULTT, SMULTB, SMULWT, SMULWB, SMUSD, SSAT16, SSAX, SSUB16, SSUB8, SXTAB, SXTAB16, SXTAH, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSAX, UHSUB16, UHSUB8, UMAAL, UQADD16, UQADD8, UQASX, UQSAX, UQSUB16, UQSUB8, USAD8, USADA8, USAT16, USAX, USUB16, USUB8, UXTAB, UXTAB16, UXTAH, UXTB16	32-bit					x	x	v7e (DSP)
VABS, VADD, VCMP, VCMPE, VCVT, VCVTR, VDIV, VLDM, VLDR, VMLA, VMLS, VMOV, VMRS, VMSR, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VPOP, VPUSH, VSQRT, VSTM, VSTR, VSUB	32-bit					SP FPU	SP FPU	32-b FP
FP- dubbel precision	32-bit						DP FPU	64-b FP

Måldator – minnesdisposition



- Minneskonfiguration – Hur mycket minne finns det och var är det placerat i adressrummet?

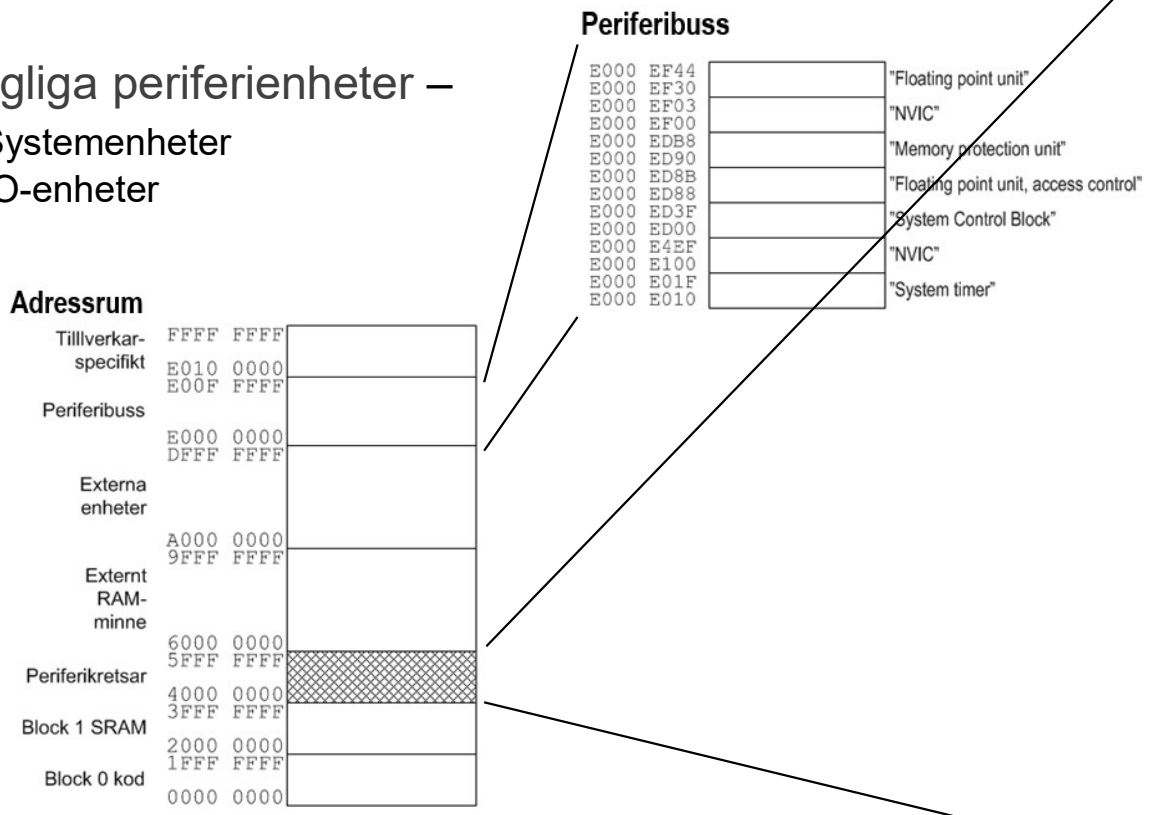


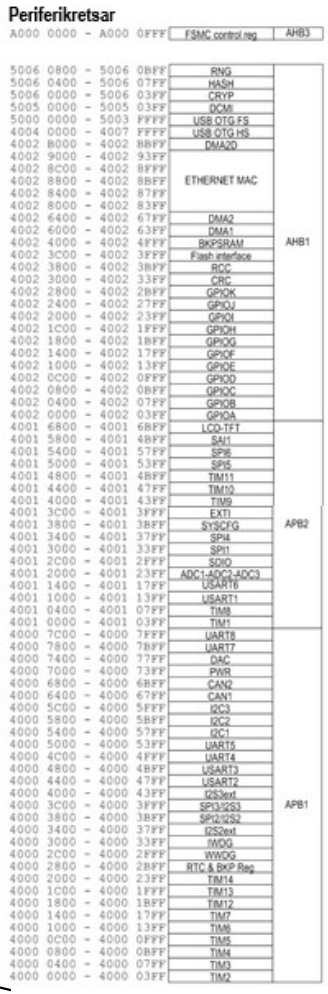


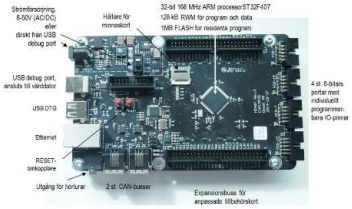
Informationen måste finnas för att vi exempelvis ska kunna implementera dynamisk minneshantering med ”malloc/free”

Måldator – periferienheter

- Tillgängliga periferienheter –
  - Systemenheter
  - IO-enheter







Informationen måste finnas för att vi exempelvis ska kunna implementera ”drivrutiner” för enheterna

Kompilatorbibliotek

gencode.c

```
int     a,b,c;
void    f( void )
{
        a = b/c;
}
```

gencodeV7.dass (arm**v7-m**)

```
LDR   R2,b
LDR   R3,c
SDIV  R3,R2,R3
LDR   R2,=a
STR   R3,[R2]
```

gencodev6.dass (arm**v6-m**)

```
LDR   R0,b
LDR   R1,c
BL    __divsi3
LDR   R3,=a
STR   R0,[R3]
```

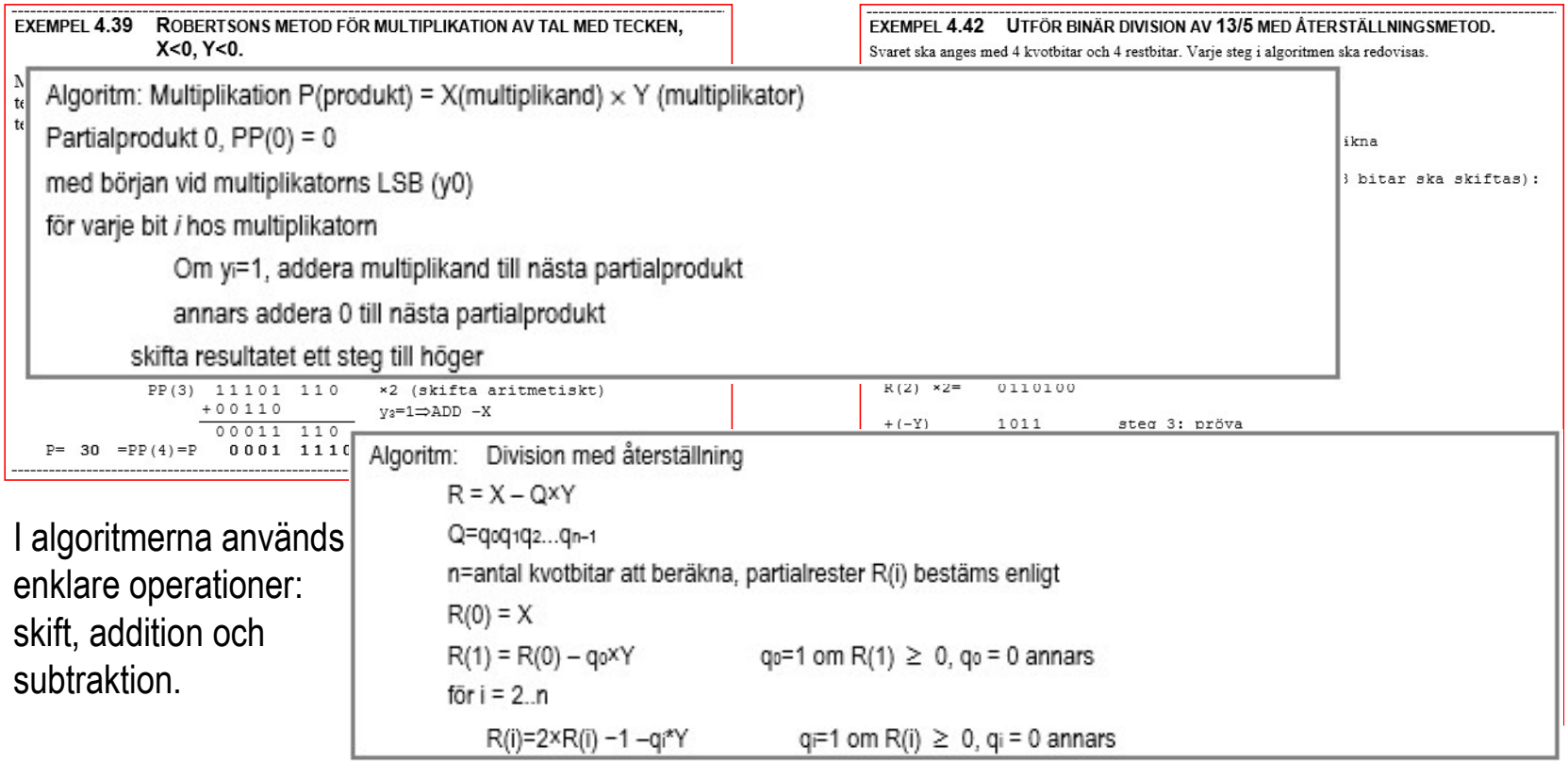
lib/gcc/arm-none-eabi/.../...libgcc.a

```
10001010011100011001010
__divsi3:
00101100001010001001001001010000
1010011100011.....
```

Funktionerna i kompilatorbiblioteket används av kompilatorn internt för kodgenerering, dvs. är ej publika och det behövs därför ingen header-fil med deklarationer.
Information om processorarkitektur måste vara konsistent vid kompilering och länkning.

Kompilatorbibliotek

I Grundläggande datorteknik har vi studerat algoritmer för multiplikation och division. Dessa kan användas för mjukvaruimpementering av operationerna.



I algoritmerna används enklare operationer: skift, addition och subtraktion.

Kompilatorbibliotek

Representation av flyttal

Ett flyttal uttrycks allmänt som:

$(-1)^S \ M \times 2^E$

där:

S (*sign*) är teckenbiten för flyttalet

S=0 anger ett positivt flyttal ty $(-1)^0 = 1$

S=1 anger ett negativt flyttal ty $(-1)^1 = -1$

M utgör talets mantissa

E utgör talets exponent.

Exponenten väljs från någon representation med inbyggt tecken. Det är värt att notera att för ett normaliserat flyttal på binär form gäller att:

$(M)_2 = 1.xxxxx$

dvs. mantissans första siffra är alltid är 1. Detta innebär att vi, då vi lagrar flyttal, kan utelämna denna siffra och på så vis åstadkomma ett kompaktare format.

Exempel: Omvandling av heltal till IEEE flyttal

Vi vill skriva talet $(2,52)_{10}\ 10^4$ som ett IEEE754-single format flyttal:

Vi har tidigare kommit fram till resultatet:

$(2,52)_{10}\ 10^4 = \quad = (1.100\ 0100\ 1110\ 000)_2 \times 2^{14}$

Mantissan ska ha totalt 24 bitar, vi ”fyller på” med nollor på slutet...

$M = (1.100\ 0100\ 1110\ 0000\ 0000\ 0000)_2$

Signifikanden F dvs. den del av mantissan som ska lagras får stryker den mest signifikanta ettan, vi har då:

$F = (100\ 0100\ 1110\ 0000\ 0000\ 0000)_2$

Exponenten i IEEE-formen uttrycks av karakteristikan E', excess(127) kod, dvs. $E' = E+127$, där E betecknar exponenten i talet vi utgår från (2^{14}) dvs. E=14 varför

$E' = 14 + 127 = 141 = (1000\ 1101)_2$

eftersom talet är positivt får vi $S = 0$. Vi sammanställer nu resultatet i 32-bitars form (*SFP*) och får slutligen:

$(2,52)_{10}\ 10^4 = (0100\ 0110\ 1100\ 0100\ 1110\ 0000\ 0000\ 0000)_{SFP}$

Representationen av heltal och flyttal är olika men de lagras med samma storlek.

```
float  f;     int i; /* båda typerna är 32 bitar */
```

Det innebär exempelvis att tilldelningen:

```
        f = (float) i;
```

ska göras enligt exemplet ovan...

Kompilatorbibliotek

```
float   f;
int     i;
    ...
    f = (float) i;
```

(armv6-m) exempelvis Cortex M3

Flaggor till kompilatorn:
`-mthumb;-march=armv6-m;-msoft-float`

ger koden:

```
        LDR     R0,i
        BL      __aeabi_i2f
        LDR     R1,=f
        STR     R0,[r1]
```

(armv7-m + floating point unit) exempelvis Cortex M4

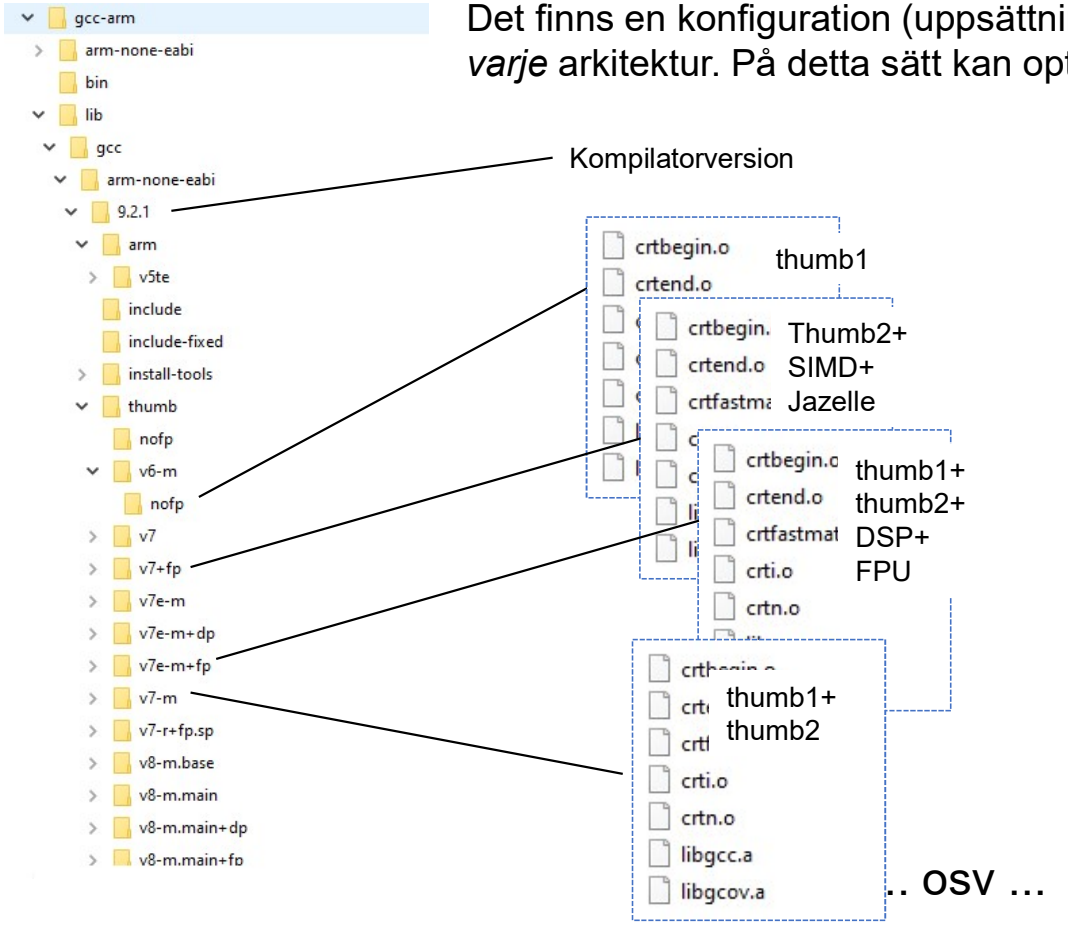
Flaggor till kompilatorn:
`-mfloat-abi=hard;mthumb;-mfpu=fpv4-sp-d16;-march=armv7-m;`

ger koden:

```
LDR             R0,i
VMOV            S15,R0
VCVT.F32.S32    S15,S15
LDR             R0,=f
VSTR.32         S15,[R0]
```

Kompilatorbibliotek - olika konfigurationer

Det finns en konfiguration (uppsättning kompilator-bibliotek och startfiler) för *varje* arkitektur. På detta sätt kan optimal kod länkas till applikationen.

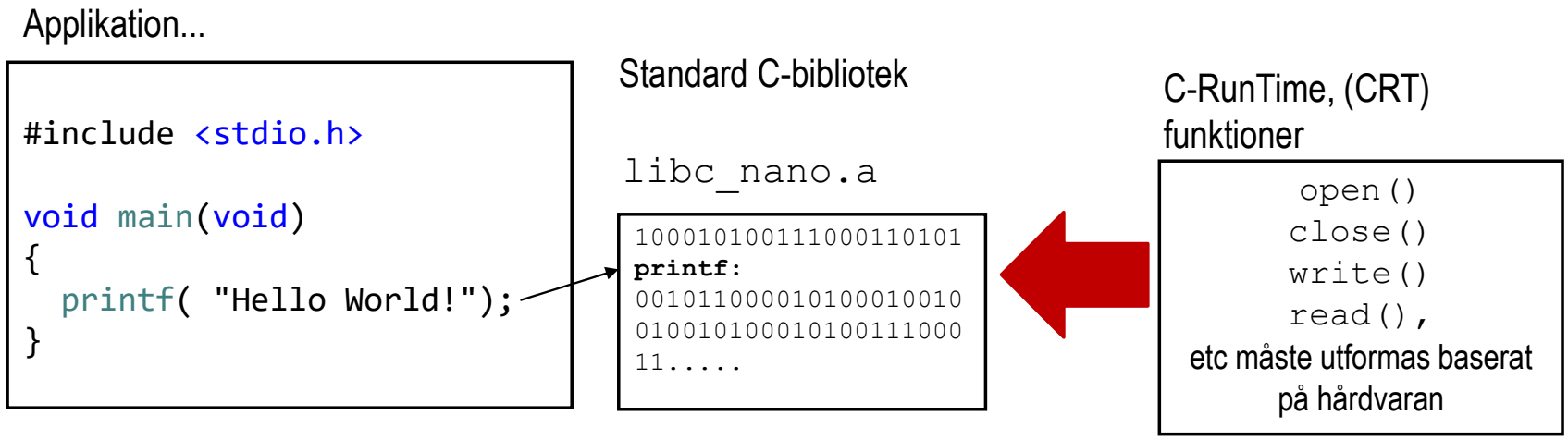


RTE - Run Time Environment
crtbegin, crtend etc..
Kallas också ”startfiles”

Libgcc, kompatibilitetskod

För att *undvika* länkning med standard filer ger man länkarflaggan **–nostartfiles**

"C-nano" - lättviktsimplementering av Standard C



Då vi länkar vår applikation upptäcker vi att en rad symboler saknas.
Detta beror på att IO-funktioner i standard C-biblioteket också länkas mot *maskinberoende* funktioner (C-RunTime), som:

`_open, _close, _lseek, _read, _write, _fstat, _isatty`

För att kunna använda IO-funktionerna måste vi då först implementera dessa maskinberoende funktioner, i form av så kallade *drivrutiner*, för MD407.

Exempel: "Device driver" keypad och ASCII-display

Datastruktur: ”Device driver”.

```
#include <sys/stat.h>

typedef struct
{
  char name[16];
  int (*init) (int);
  void (*deinit) (int);
  int (*fstat)(struct stat *st);
  int (*isatty)(void);
  int (*open)(const char name,int flags,int mode);
  int (*close)(void);
  int (*lseek)(int ptr, int dir);
  int (*write)(char *ptr, int len);
  int (*read)(char *ptr, int len);
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;
```

```
static DEV_DRIVER_DESC KeyPad =
{
  {"Keypad"},
  keypad_init,
  keypad_deinit,
  0,
  0,
  0,
  0,
  0,
  0,
  keypad_read
};
```

```
static DEV_DRIVER_DESC AsciiDisplay =
{
  {"AsciiDisplay"},
  asciidisplay_init,
  asciidisplay_deinit,
  0,
  0,
  0,
  0,
  0,
  asciidisplay_write,
  0
};
```

Exempel: Tabell med "Device drivers"

```
#define MAX_FILENO              5

PDEV_DRIVER_DESC device_table[MAX_FILENO] =
{
          &StdIn,
          &StdOut,
          &StdErr,
          &KeyPad,
          &AsciiDisplay,
};

void crt_init() {
  …
  for( int i = 0; i < MAX_DEVICE; i++ )
  {
    fd = device_table[i];
    if( fd->init != 0)
      (void) fd->init( 0 );
   }
}
```

```
static DEV_DRIVER_DESC StdIn =
{
  {"stdin"},
  usart_init,
  usart_deinit,
  0,
  usart_isatty,
  0,
  0,
  0,
  0,
  usart_read
};
```

```
static DEV_DRIVER_DESC StdOut =
{
  {"stdin"},
  0,
  usart_deinit,
  0,
  usart_isatty,
  0,
  0,
  0,
  usart_write,
  0
};
```

"C-run time" - lättviktsimplementering för MD407

_fstat, returnera information om en öppen fil.

```
#include <sys/stat.h>
int _fstat(int file, struct stat *st) {
 st->st_mode = S_IFCHR;
 return 0;
}
```

`struct stat` och `S_IFCHR` deklareras i `sys/stat.h`

S_IFCHR, anger att detta är en tecken orienterad fil, andra exempel är:
`S_IFDIR`, filen är ett bibliotek.
`S_IFBLK`, filen är block-orienterad, (typiskt på disk) osv...
Vår implementering kommer endast att stödja de teckenorienterade enheter.

_isatty ska returnera 1 om filen är av terminal-typ. `stdin`, `stdout` och `stderr`, kan normal sett dirigeras om till blockorienterade enheter, dock inte i vår implementering. `isatty` implementeras därför, men enbart för `stdin`, `stdout` och `stderr`.

```
int _isatty(int file) { return 1; }
```

"C-run time" - lättviktsimplementering för MD407

`_open`, öppna en fil för läsning och/eller skrivning.
Ej tillämpbart, vi har inget filsystem, våra enheter öppnas (`init`) av runtime-systemet. Det räcker då att vår implementering returnerar en felkod om `open` anropas.

```
int _open(const char *name, int flags, int mode) { return -1; }
```

`_close`, stäng en fil som är öppen för läsning och/eller skrivning.
Ej tillämpbart, våra enheter stängs av runtime-systemet (`deinit`).

```
int _close(int file) { return -1; }
```

`_lseek`, sök till position i en fil som är öppen för läsning och/eller skrivning.
Måste vara en blockorienterad fil, ej tillämpbart för teckenorienterade enheter.

```
int _lseek(int file, int ptr, int dir) { return 0; }
```

"C-run time" - lättviktsimplementering för MD407

`_write`, skriv till en öppen fil eller enhet.
Funktionen ska dirigera operationen till enhetens write-funktion om en sådan finns.

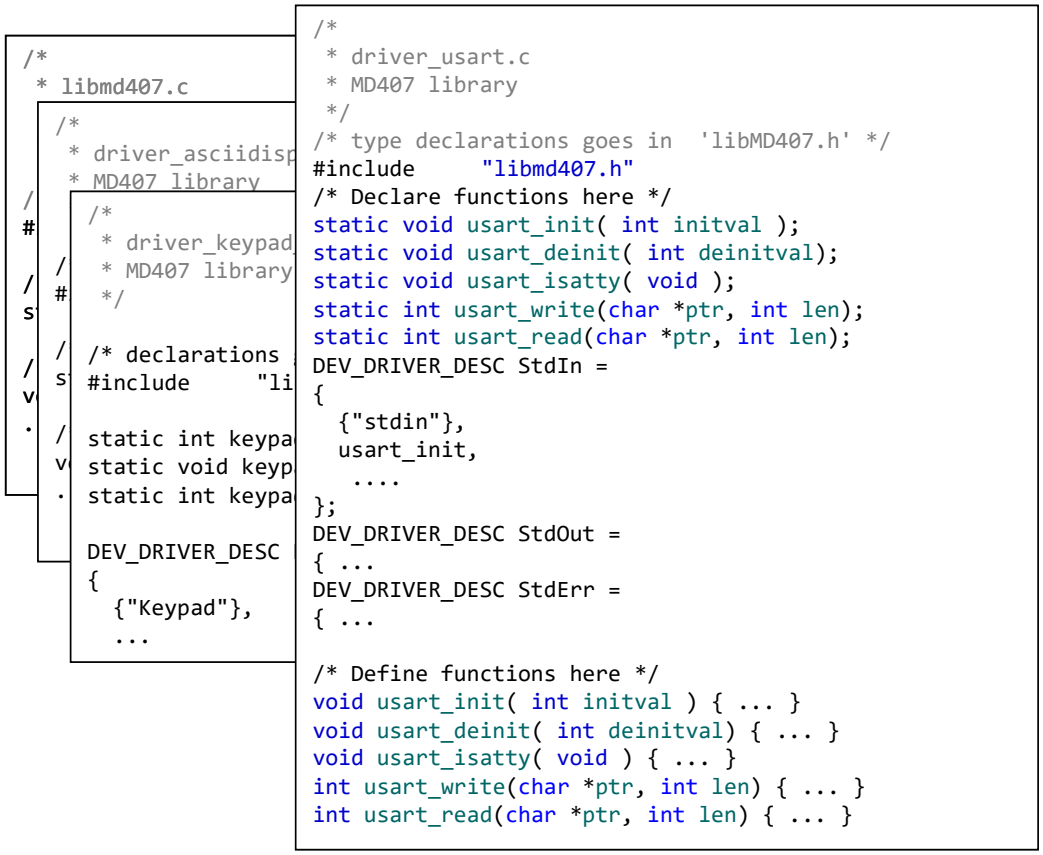
```
int _write(int file, char *ptr, int len) {
  PDEV_DRIVER_DESC drvr;
  drvr = device_table[file];
  if(drvr == 0)
    return 0;
  return drvr->write(ptr,len);
}
```

`_read`, läs från en öppen fil eller enhet.
Funktionen ska dirigera operationen till enhetens read-funktion om en sådan finns.

```
int _read(int file, char *ptr, int len) {
  PDEV_DRIVER_DESC drvr;
  drvr = device_table[file];
  if(drvr == 0)
    return 0;
  return drvr->read(ptr,len);
}
```

Skapa run-time bibliotek för MD407

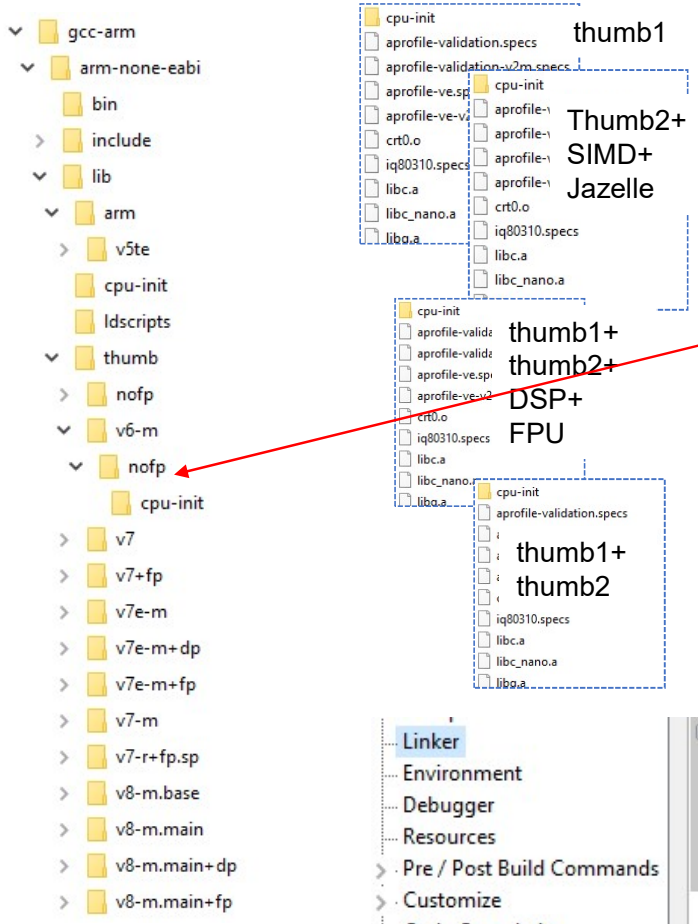
Under laboration 6 ska du skapa ett programbibliotek `libmd407.a` som kompletterar med de hårdvaruberoende run-time funktionerna



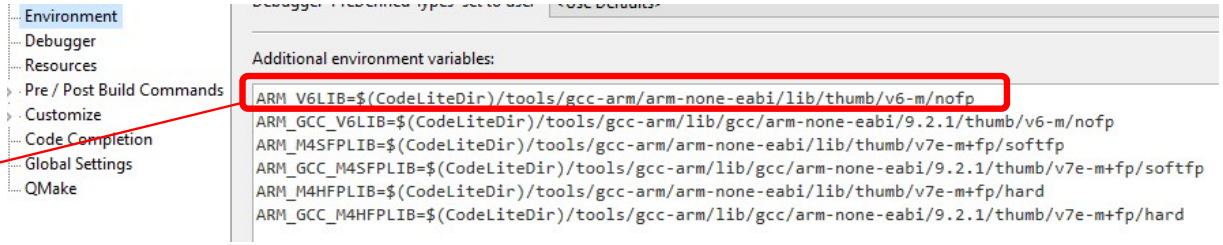
```c
/*
 * driver_usart.c
 * MD407 library
 */
/* type declarations goes in  'libMD407.h' */
#include     "libmd407.h"
/* Declare functions here */
static void usart_init( int initval );
static void usart_deinit( int deinitval);
static void usart_isatty( void );
static int usart_write(char *ptr, int len);
static int usart_read(char *ptr, int len);
DEV_DRIVER_DESC StdIn =
{
  {"stdin"},
  usart_init,
   ....
};
DEV_DRIVER_DESC StdOut =
{ ...
DEV_DRIVER_DESC StdErr =
{ ...

/* Define functions here */
void usart_init( int initval ) { ... }
void usart_deinit( int deinitval) { ... }
void usart_isatty( void ) { ... }
int usart_write(char *ptr, int len) { ... }
int usart_read(char *ptr, int len) { ... }
```

```c
/*
* libmd407.h
* Declaration of library functions, constants etc...
*/
#ifndef      _LIBMD407_H
#define      _LIBMD407_H
#include     <stdio.h>
#include     <errno.h>
#include     <sys/stat.h>
#include     <unistd.h>

/* Type definitions */
typedef struct
{
 ...
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;

/* Constants */
#define MAX_FILENO  5
/* Constant defined by linker */
extern char __heap_low;
extern char __heap_top;
extern char __bss_start__;
extern char __bss_end__;

/* Library defined functions */
void crt_init( void );
void crt_deinit( void );

#endif       /* LIBMD407_H */
```

C-bibliotek – olika konfigurationer

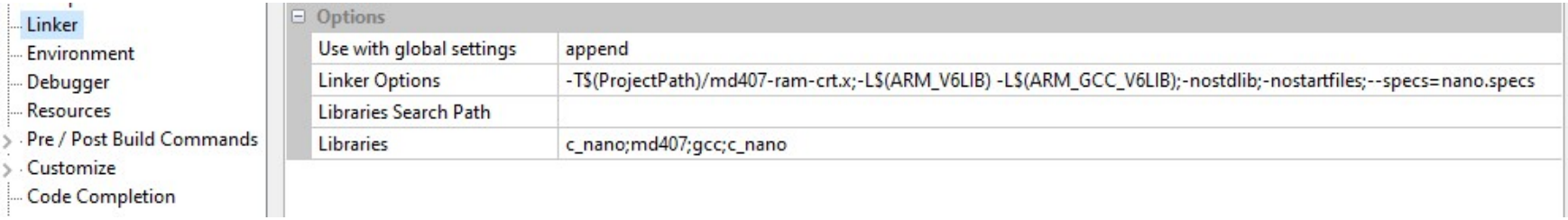


På samma sätt finns det en konfiguration (uppsättning C-bibliotek) för *varje* arkitektur. Vi använder miljövariabler för att precisera biblioteken:

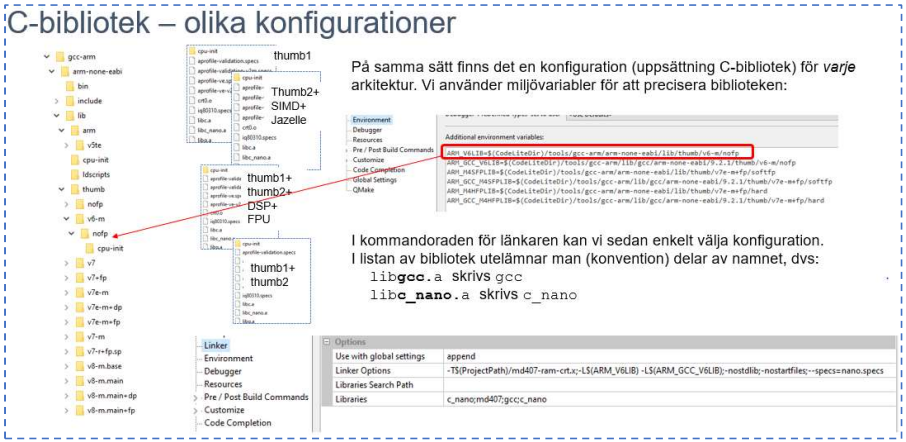


I kommandoraden för länkaren kan vi sedan enkelt välja konfiguration.
I listan av bibliotek utelämnar man (konvention) delar av namnet, dvs:

`lib`**`gcc`**`.a` skrivs `gcc`
`lib`**`c_nano`**`.a` skrivs `c_nano`



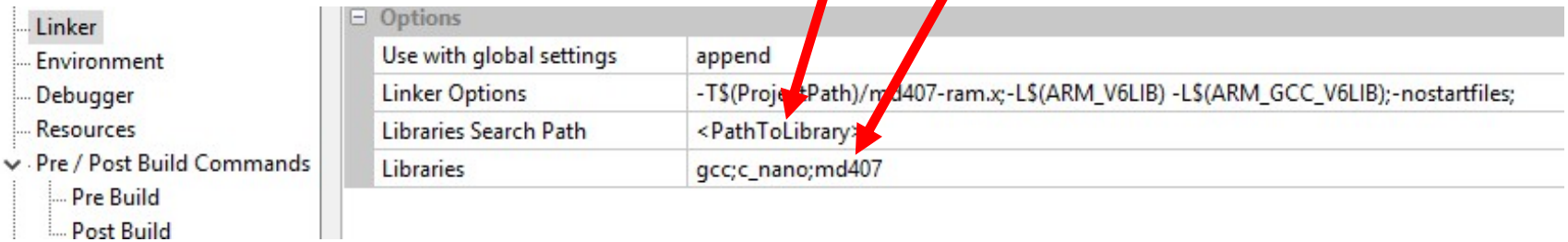
`libmd407.a`, installation och användning



Sökväg till det nya biblioteket.

Namn på nytt bibliotek.

Man kan välja att installera biblioteket i en befintlig sökväg, eller att skapa en ny. I vilket fall, måste man lägga till programbiblioteket i listan av bibliotek:



Demonstration: Användning av biblioteket