Embedded Systems

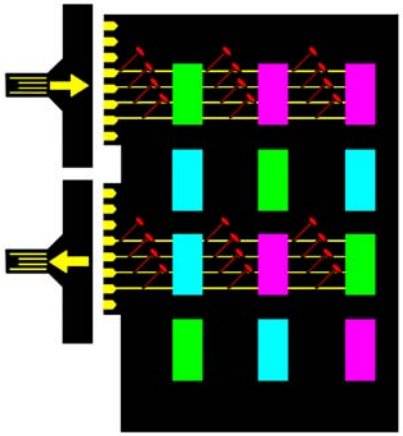
Prof. Myung-Eui Lee (F-102)
melee@kut.ac.kr
한국기술교육대학교
korea University of Technology and Education

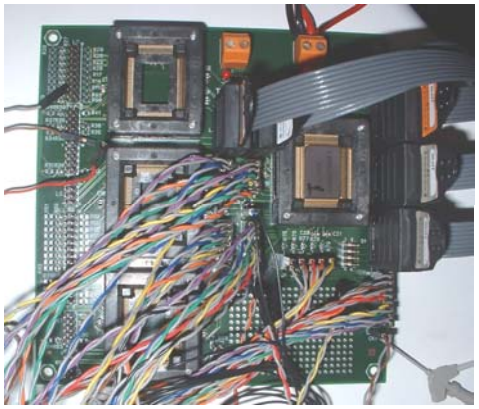
JTAG

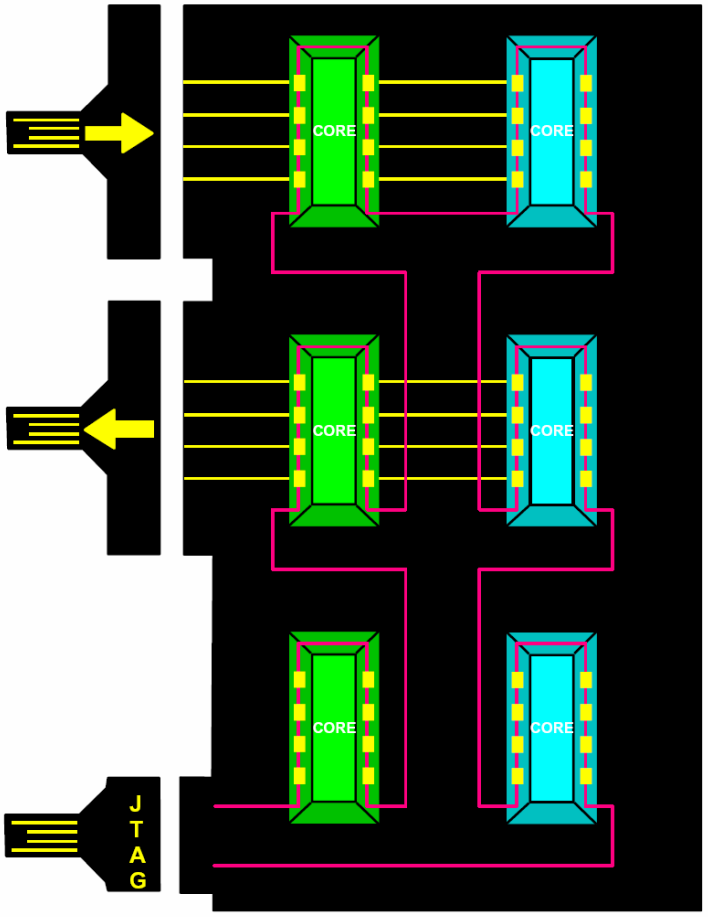
- **JTAG (Joint Test Action Group)**
  - **Boundary scan = IEEE Standard 1149 = JTAG**
  - **IEEE Standard 1149.1**
    - Standard Test Access Port and Boundary-Scan Architecture
  - **JTAG committee was formed by test professionals within Philips, BT, GEC and others in 1990.**
  - **JTAG was standardized in 1990 as the IEEE Std. 1149.1-1990.**
  - **Objectives**
    - In circuit test for the surface mount designs and printed circuit boards using boundary scan
    - Test interconnections between Integrated Circuits (ICs) installed on boards (modules)
      - → could not so easily be probed by testers.
  - **Embedding test cells within a device**
    - Examine internal board states without physical contact
    - Program PLDs and Flash memories

JTAG

- **Conventional Board Test vs Boundary Scan Idea**





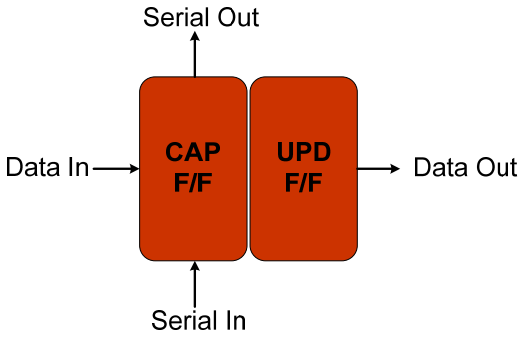


JTAG

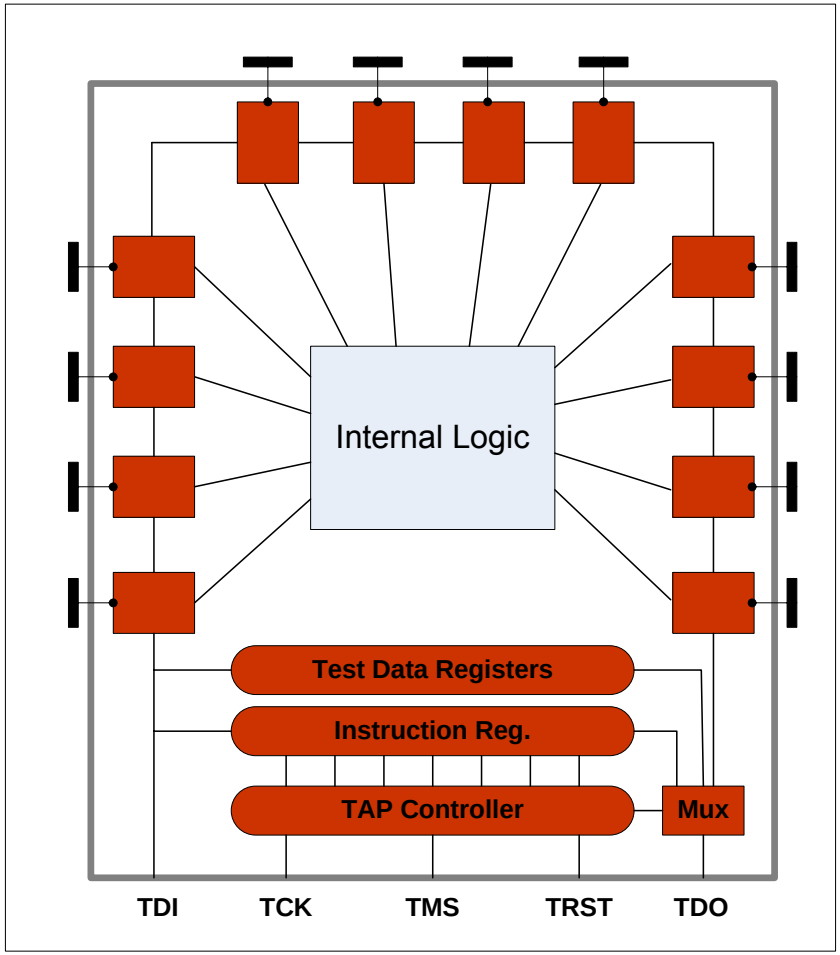
- **Basic Architecture**
  - **Test Access Port (TAP) : JTAG interface (4/5 pin)**
    - TDI, TDO, TCK, TMS, TRST (optional)
  - **Instruction Register**
    - receives an instruction through the TDI, decodes it, and selects the appropriate data register
    - sets the mode of operation for one or more data registers
  - **Data Registers**
    - Several different data registers can be built into boundary-scan components
      - → Boundary scan reg./ Bypass reg./ Dev ID reg./ User defined reg.,,,,
    - Two Data Registers are always required to be present on a 1149.1 component:
      - → **Boundary-Scan Register** : Boundary Scan Cell (BSC)
      - → **Bypass Register** : short-cut scan path through devices that are not involved in the test (Bypass instruction)
  - **TAP Controller**
    - generates internal control signals, 16-state finite state machine
    - 4 groups (16 states) : Reset, BIST, Data reg. update, Inst. Reg. update

JTAG

- **JTAG Components**
  - » Test Access Port (TAP)
  - » Instruction Register
  - » Data Registers :
    Boundary Scan Cell (BSC)
    Bypass Register
  - » TAP Controller

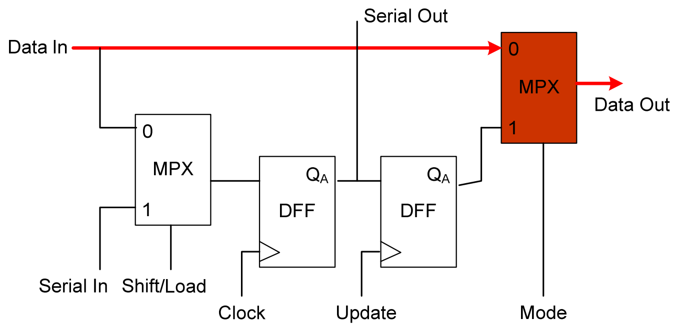


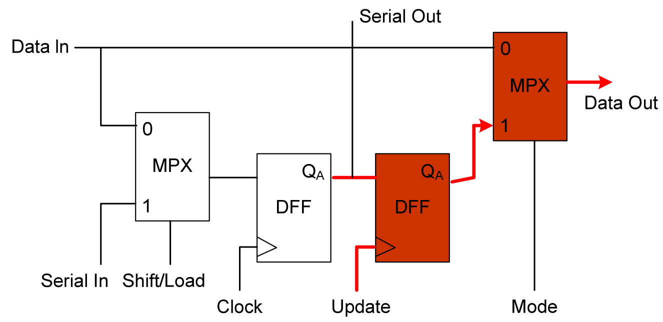
BSC

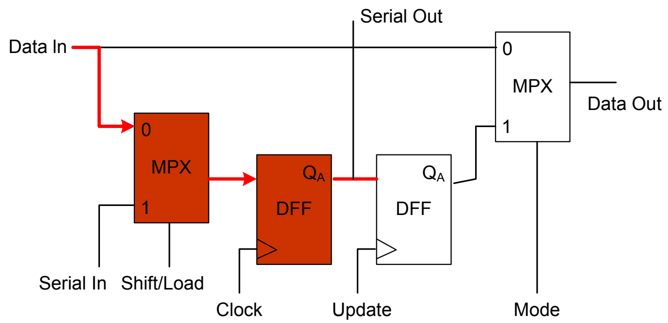


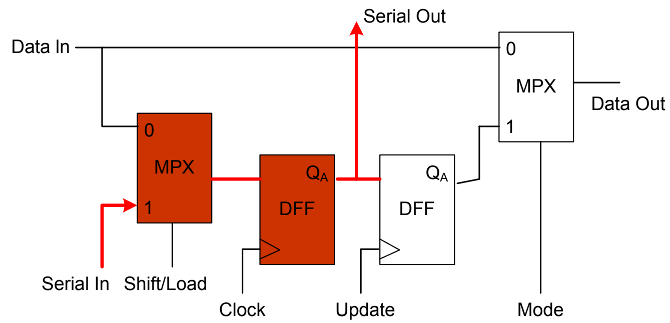
- **BSC 4 different functional modes**

  1. Normal mode : Mode = 0
  2. Update mode : Mode = 1
  3. Capture mode : Shift/Load = 0
  4. Scan Mode : Shift/Load = 1



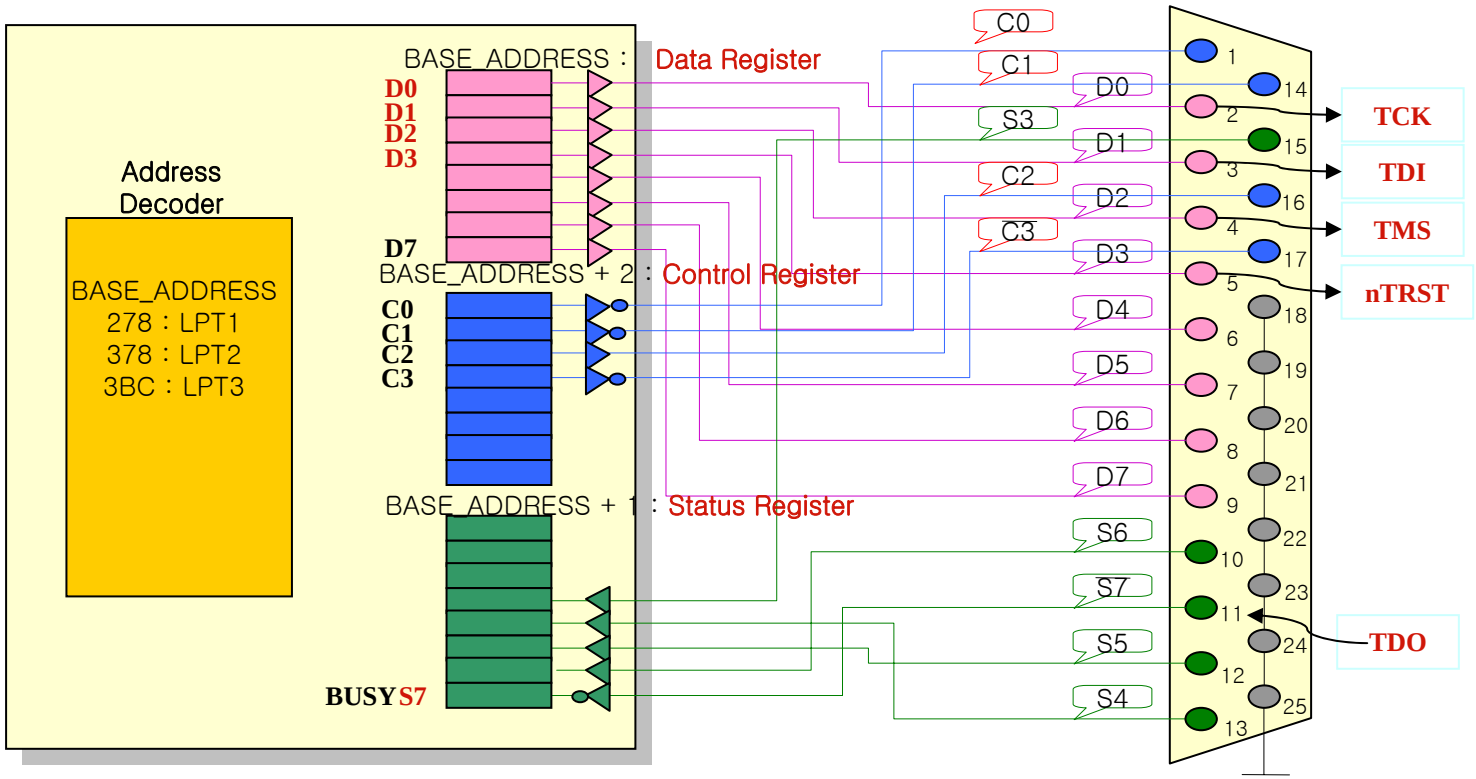






- **Host PC Interface : JTAG Cable** (printer or usb)

Host (PC) Parallel Port



TCK	D0	LPT pin 2
TDI	D1	LPT pin 3
TMS	D2	LPT pin 4
nTRST	D3	LPT pin 5
TDO	BUSY	LPT pin 11

Data register

Control register

Status register

Debug

- **Challenges**
  - Target system may be hard to observe
  - Target may be hard to control
  - May be hard to generate realistic inputs
  - Setup sequence may be complex
- **Tools**
  - Logic Analyzers
  - Software Debuggers
    - Monitor Program
    - Breakpoint Program
  - In-circuit Emulators (ICE)
  - ARM's EmbeddedICE
  - IDE (Integrated Development Environment)

Debug

- **Logic Analyzers**
  - Used for sampling many different signals simultaneously, and display 0, 1, or changing values for each.
  - Only offer an historical view of the action of code
  - User cannot change variables or jump to different parts of the program
  - Useful to measure time intervals
  - Fixed amount of buffer memory
    - → relates to the depth of the time window for acquiring system execution history in real-time (measure in number of samples it can hold).
    - → the deeper the memory, the more data you have to analyze to find the cause of the problem.
- **Software Debuggers**
  - Monitor
    - → A monitor program residing on the target provides basic debugging functions
    - → The need to have the monitor in ROM on the target system is a significant problem
      - » Must either be removed from the final product
      - » Or left in ROM at extra cost

Debug

- Break point
  - → A breakpoint allows the user to stop execution, examine system state, and change state
  - → Replace the breakpoint instruction with a subroutine call to the monitor program

» In-circuit Emulators (ICE)

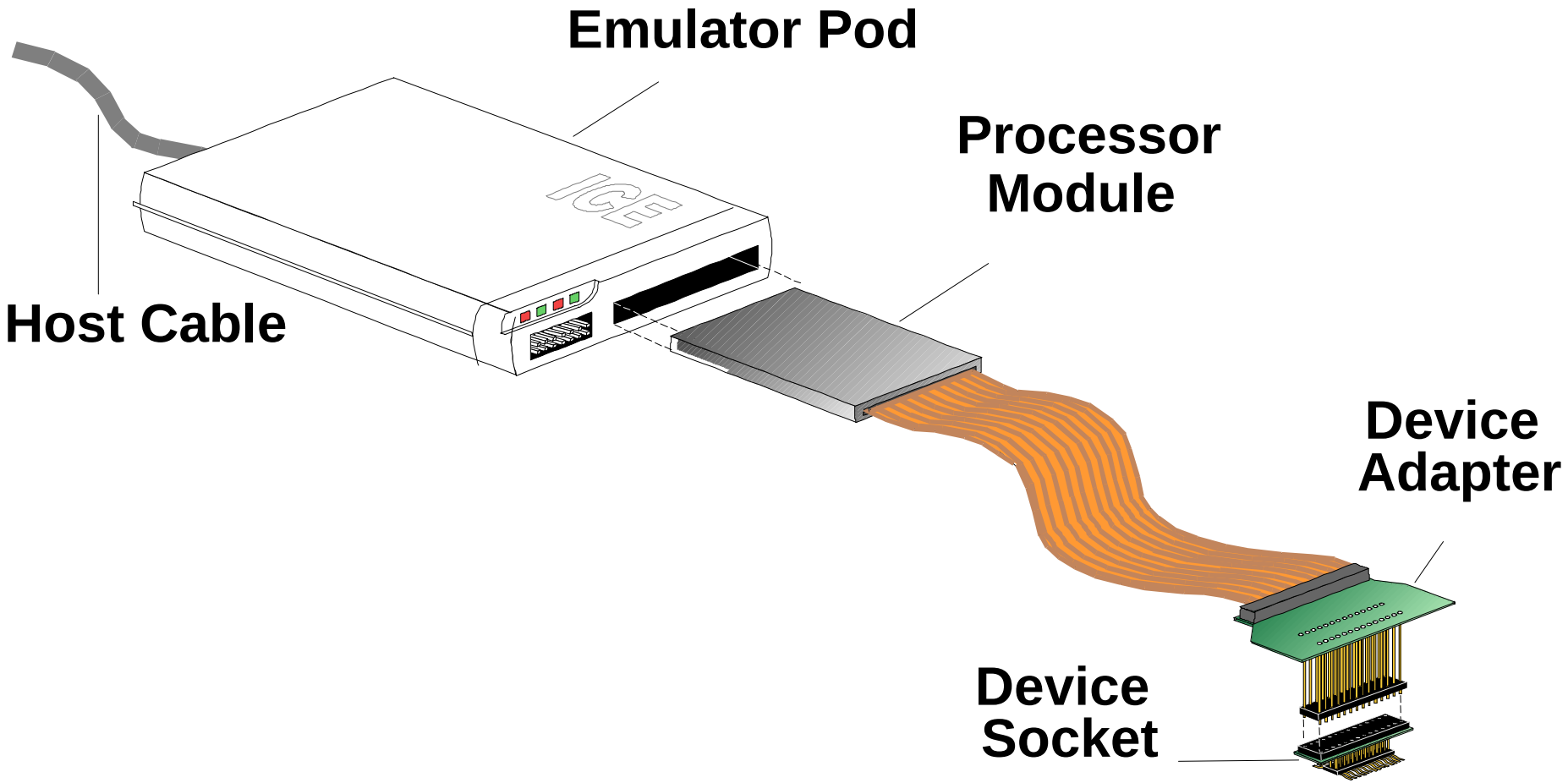
- Interface
  - → Host PC : RS232, Parallel Port (Printer), USB, LAN (Ethernet)
  - → Target : CPU replacement, JTAG, Proprietary
- A microprocessor in-circuit emulator is a specially-instrumented microprocessor
- Allows you to stop execution, examine CPU state, modify registers
- ICE is a dedicated hardware for a particular microprocessor (different processor/different processor module or pod)
- ICE can be expensive

» ARM's EmbeddedICE

- JTAG-based debugging channel for ARM microprocessors
- An EmbeddedICE-compatible ARM core with a boundary scan interface and debug enhancements

Debug

- **ICE Example**



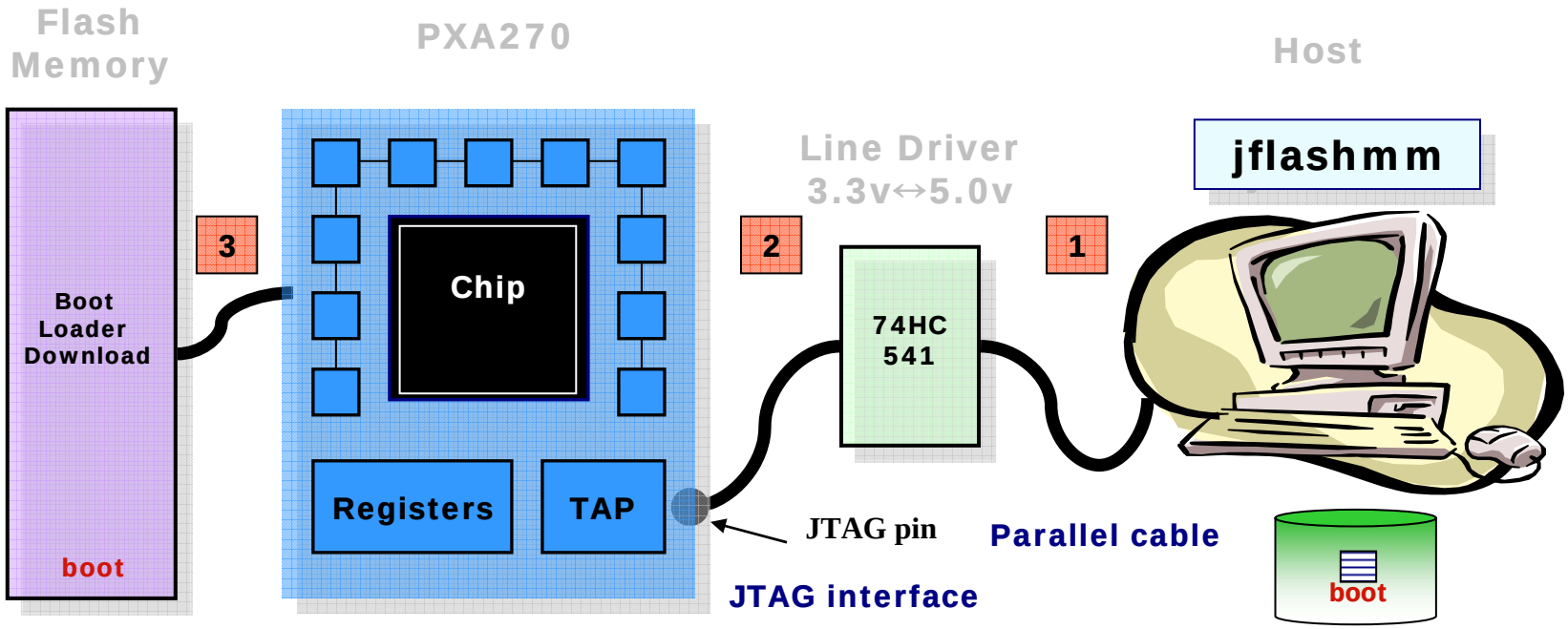
Debug

- An external EmbeddedICE Interface Box which links the host development machine with the debug compatible ARM
- No target resources or special hardware
- Low cost solution that does not require dedicated ICE
- Debug can be performed at full processor speed
- Full host system access including screen, keyboard and storage for the target
- Requires no extra communication channel to debug
- Host software development and debug tools -> RealView Multi ICE

» **IDE (Integrated Development Environment)**

Flash Fusing

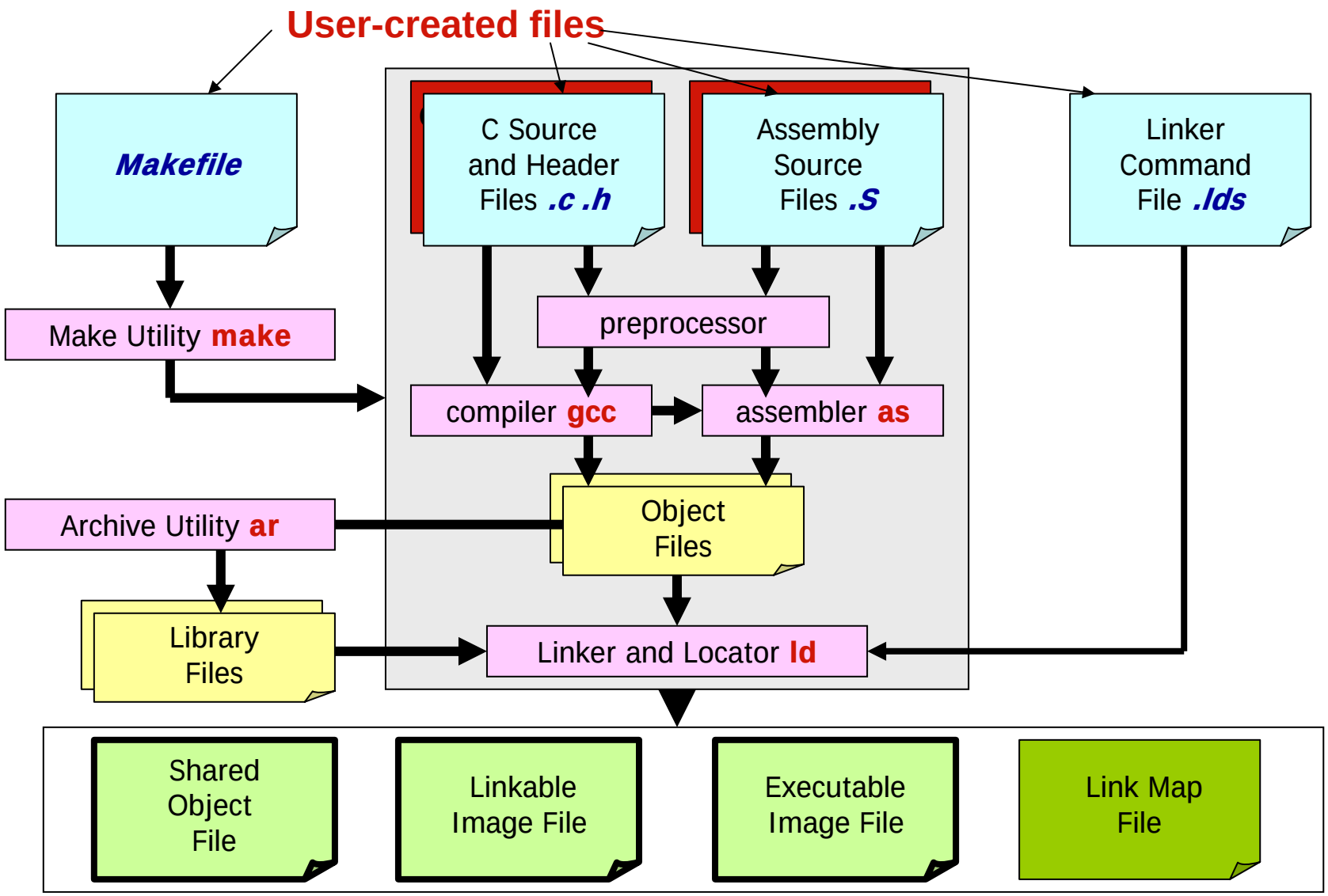
- **Application S/W** : /pxa270/jflashmm/jflashmm-5.01.007-h2
- **Boot loader source** : /pxa270/bootloader/bboot-1.0.1
- **Boot loader fusing**
  - *cd /pxa270/bootloader/bboot-1.0.1*
  - *make clean, make*
  - *./jflashmm pxa27x32.dat boot*



Flash Fusing

- **Using tftp and flash at boot monitor**
  - cd /pxa270/bootloader/bboot-1.0.1/src (host)
  - vi main.c <- logo change
  - cd ..
  - make
  - cp boot /tftpboot
  - tftp boot loader (target)
  - flash loader

Overview of source translation



GNU toolchains

- **GNU toolchains**
  - **GNU Compiler : gcc**
  - **GNU C Library**
  - **GNU Binary Utilities**
    - ld, as, ar, gdb, make, objcopy, objdump, readelf, ...
- **GNU Binutils**
  - ***make*** **– flexible rule-based project builder which helps maintaining large software systems**
  - ***ar*** **– create and maintain *archives (libraries)***
  - ***objdump*** **– extract and display information from an object file**
  - ***objcopy*** **–translate object files to a different format**

gcc

- **gcc hello.c**
  - compile `hello.c`, produce executable `a.out`
- **gcc -o hello hello.c** (gcc hello.c -o hello)
  - compile `hello.c`, produce executable `hello`
- **gcc -o hello hello.c func-a.c func-b.c...**
  - compile `hello.c func-a.c` and `func-b.c`, ... produce executable `hello`
- **gcc -c hello.c**

  gcc -c funcs.c

  gcc -o hello hello.o funcs.o
  - produce an object file to be linked in later to an executable

gcc

- **Basic options**
  - `-g` : include debugging symbols in the output
  - `-l<name>` : include a library called `libname.a`
  - `-I<path>` : look for include files in this directory
  - `-L<path>` : look for library files in this directory
- **Define**
  - set preprocessor defines on the command line

```
gcc -DDEBUG -o prog prog.c
```

  - conditional parts based on define

```
#ifdef DEBUG
printf("value of var is %d", var);
#endif
```

make

- **Reads a file called makefile or Makefile, which contain rules for building a "target"**
- **`make` automates this process**
  - **Type commands to compile all the files correctly each time**
  - **Keep track of which files have been changed**
  - **Keep track of files' dependencies on other files**
- **Types of lines in Makefiles**
  - **Dependencies**
  - **Commands**
  - **Macro assignments**
  - **Special variables in Command line**
  - **Suffix rules**
  - **Comments (begin with a '#')**

make

- **Dependencies / Commands**
  - Specify a target and a list of prerequisites (optional) for that target
  - Command lines must start with a TAB

```
target : prereq1 prereq2 prereq3 …
        command1
        command2
```

  -
```
prog : main.c func-a.c func-b.c
   gcc -o prog main.c func-a.c func-b.c
```

  - More detail

```
prog : main.o func-a.o func-b.o
       gcc -o prog main.o func-a.o func-b.o
main.o : main.c
       gcc -c main.c
func-a.o : func-a.c
       gcc -c func-a.c
func-b.o : func-b.c
       gcc -c func-b.c
```

make

- **Macros**
  - represent other text in a Makefile
  - Assignment : `MACRONAME = macro value`
  - Usage : `${MACRONAME}`

```
OBJS = main.o func-a.o func-b.o
CC = /usr/bin/gcc
prog : ${OBJS}
	${CC} -o $@ ${OBJS}
main.o : main.c
	${CC} -c $?
func-a.o : func-a.c
	${CC} -c $?
func-b.o : func-b.c
	${CC} -c $?
```

make

- **build-in macros** : make -p
  - CC, AR, AS, LD, CXX
  - CFLAGS, CXXFLAGS - compiling flags
  - LDFLAGS, ASFLAGS - linking, assembler flags
  - .SUFFIXES - store suffixes (make –p | grep .SUFFIXES)

- **Special variables in commands:**
  - $@ represents the target
  - $? represents the prerequisite that are newer than target
  - $< represents the prerequisite file name in a suffix rule
  - $$ represents the process number of current shell

make

- **Suffix rules**
  - still tedious to specifically tell `make` how to build each .o file from a .c file
  - default suffix rule turns .c files into .o files by running the command: `gcc -c $<`
  - .SUFFIXES: .o .c - store suffixes in the .SUFFIXES macro
  - .c.o: (make –p | grep .c.o)

    gcc -c $< - rule for generating *.o from *.c
  - %.o: %.c

    gcc -c $< - %.c means every file with suffix .c
  - final Makefile using suffix

```
OBJS = main.o func-a.o func-b.o
CC = /usr/bin/gcc
prog : ${OBJS}
       ${CC} -o $@ ${OBJS}
%.o: %.c
       ${CC} -c $<
```

make

- **Invoking make**
  - `make` : builds the first target in the file
  - `make` *`target :`* builds target
- **Make options:**
  - `-n` : don't run the commands, just list them
  - `-f` *`file :`* use file instead of Makefile or makefile
- ***clean*** **node**
  - usually to remove unnecessary files

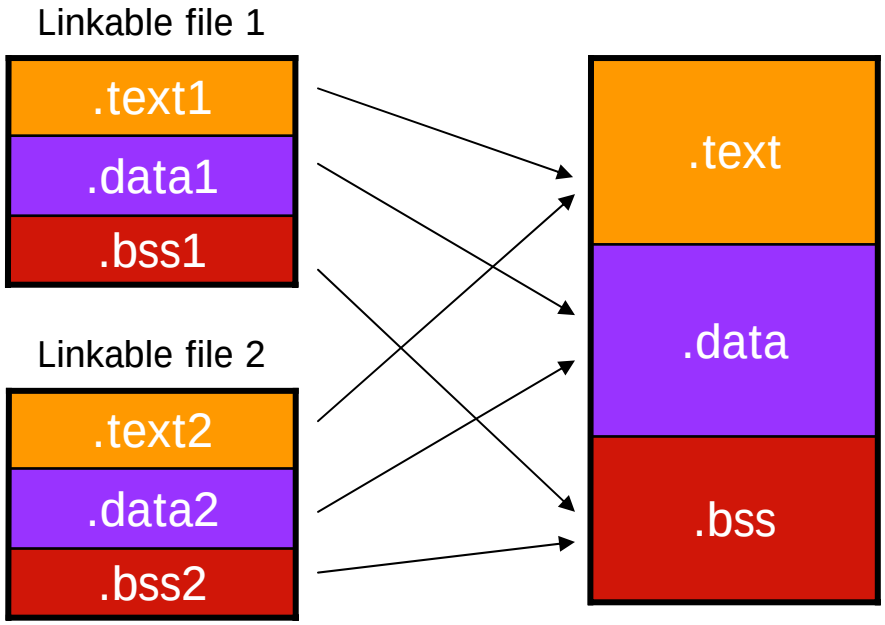
    clean:

    *rm -f *.o*
  - `make clean`
- ***all*** **node**
  - Include a target to build multiple programs

    all: prog1 prog2 prog3
  - by default make command would execute ***all*** node

Linker

- **Object file : gcc –c hello.c -> hello.o**
- **The linker combines *object modules* which may be available in a variety of file formats**
  - » **Executable and Linking Format (ELF) : de facto standard**
  - » **Common Object File Format (COFF) : former binary format**
  - » **Debug With Arbitrary Record Format (DWARF) : Debug Information Format**



Linker

- **The most important sections**
  - ***.text*** **- contains code and constants**
  - ***.data*** **- initialized data**
  - ***.bss*** **- un-initialized data** (block starting at symbol, block started by symbol, block start symbol, blank storage space, block storage segment)
    - Give a name to the beginning of a block of data.
  - ***.rodata*** **- ROM variables**
  - ***.common*** **- shared overlayed data sections**
  - ***.vector*** **- interrupt vector table**
  - ***.got*** **– global offset table**
    - stores the final (absolute) location of a function calls symbol, used in dynamically linked code.
    - located in the .got section of an ELF executable or shared object.
  - ***.comment*** **- documenting comments**

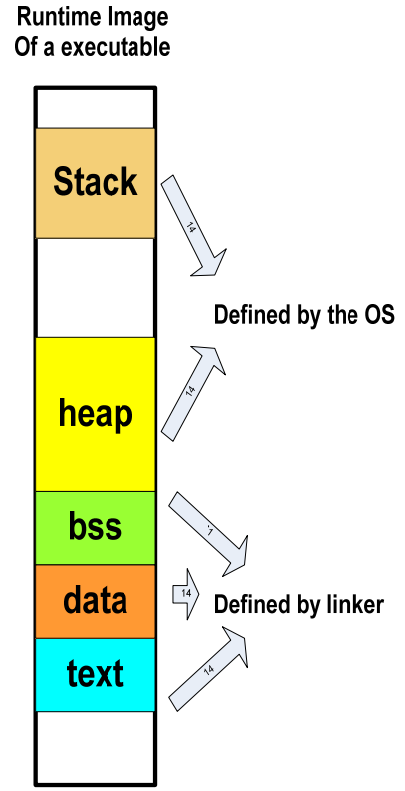
segment

- **segments : Segmented memory**
  - » **a technique used in computer hardware to divide memory up into smaller, more manageable units.**
  - » **memory segments are managed by a memory management unit (MMU), under operating system control.**

Program's Address Space

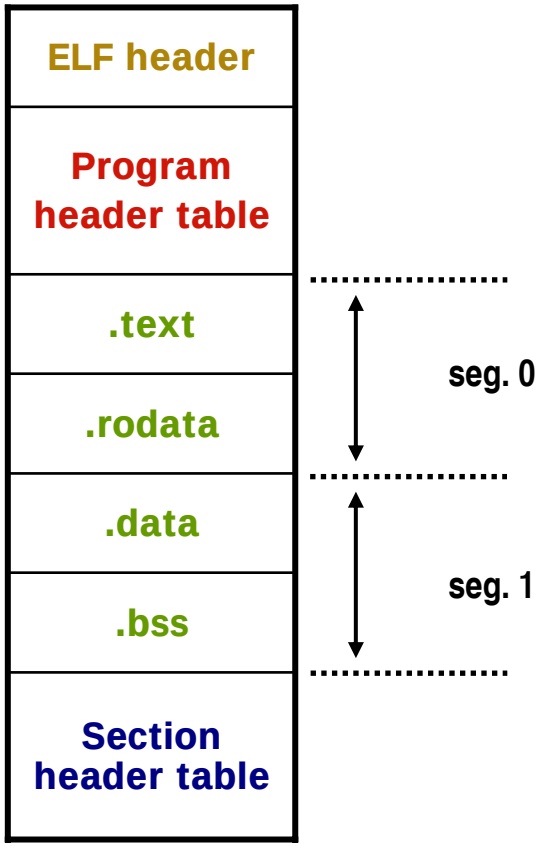
- A program's address space is defined by linker and operating system.

			.text	.rodata	.data	.bss	stack
global	static	initialized			V		
		non-initialized				V	
	non-static	initialized			V		
		non-initialized				V	
	const			V			
local	static	initialized			V		
		non-initialized				V	
	non-static	initialized					V
		non-initialized					V
	const		V				
immediate value			V				



ELF file

- **Four major parts in an ELF file**
  - **ELF header**
    - Virtual Memory Entry Point
    - Program header table file offset
    - Section header table file offset
  - **Program header**
    - Describe segments directly related to program loading
    - Program header is for ELF loader in Linux kernel
  - **Section header**
    - Describe contents of the file
    - Section header is for linker
  - **Data itself**
    - .text, .rodata, .data, .bss

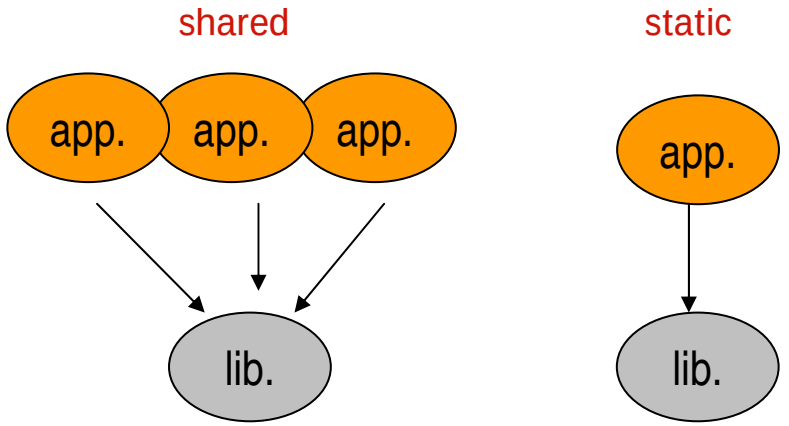


ELF file

- **3 types of ELF**
  - » **relocatable (*.o) for linker**
  - » **executable (*.exe) for loader : running**
  - » **shared object (*.so) : dynamic linking**

	Program header	Section header
relocatable		V
executable	V	V
Shared object	V	V

- **Shared / Static Library**
  - » **shared : .so**
    - malloc(), mfree()
    - dll : dynamic linking lib.
  - » **static : .a**
    - ***ar –t /usr/lib/libc.a*** : archive manager



ELF header

- **/usr/src/linux-2.4.20-8/include/linux/elf.h**

```
typedef struct elf64_hdr {
unsigned char e_ident[16];          /* ELF "magic number" */
Elf64_Half e_type;
Elf64_Half e_machine;
Elf64_Word e_version;
Elf64_Addr e_entry;                 /* Entry point virtual address */
Elf64_Off e_phoff;                  /* Program header table file offset */
Elf64_Off e_shoff;                  /* Section header table file offset */
Elf64_Word e_flags;
Elf64_Half e_ehsize;
Elf64_Half e_phentsize;
Elf64_Half e_phnum;
Elf64_Half e_shentsize;
Elf64_Half e_shnum;
Elf64_Half e_shstrndx;
} Elf64_Ehdr;
```

ELF Program header

- **/usr/src/linux-2.4.20-8/include/linux/elf.h**

```
typedef struct elf64_phdr {
  Elf64_Word p_type;
  Elf64_Word p_flags;
  Elf64_Off p_offset;          /* Segment file offset */
  Elf64_Addr p_vaddr;          /* Segment virtual address */
  Elf64_Addr p_paddr;          /* Segment physical address */
  Elf64_Xword p_filesz;        /* Segment size in file */
  Elf64_Xword p_memsz;         /* Segment size in memory */
  Elf64_Xword p_align;         /* Segment alignment, file & memory */
} Elf64_Phdr;
```

- **Program header**
  - » ***objdump –p hello-x86***

ELF Section header

- **/usr/src/linux-2.4.20-8/include/linux/elf.h**

```
typedef struct elf64_shdr {
  Elf64_Word sh_name;          /* Section name, index in string tbl */
  Elf64_Word sh_type;          /* Type of section */
  Elf64_Xword sh_flags;        /* Miscellaneous section attributes */
  Elf64_Addr sh_addr;          /* Section virtual addr at execution */
  Elf64_Off sh_offset;         /* Section file offset */
  Elf64_Xword sh_size;         /* Size of section in bytes */
  Elf64_Word sh_link;          /* Index of another section */
  Elf64_Word sh_info;          /* Additional section information */
  Elf64_Xword sh_addralign;    /* Section alignment */
  Elf64_Xword sh_entsize;      /* Entry size if section holds table */
} Elf64_Shdr;
```

- **Section Information**
  - » ***objdump –h hello.o***

Section Information

- **LMA / VMA**
  - **LMA (load memory address): the address at which a section will be loaded**
  - **VMA (virtual memory address): the runtime address of a section**
  - **In most cases the two addresses will be the same.**
  - **An example of different**
    - a data section is loaded into ROM, and then copied into RAM when the program starts up

Linker script

- **Linker script**
  - The originally UNIX based *GNU gcc compiler* has been ported to a large number of environments
  - gcc does not define memory models but instead works with *linker script* files
  - *Linker script* files describe the which sections will be part of the final program and in which order they appear
  - Define additional symbols for use in your code
  - built-in default linker script which commonly works for the majority of simple applications
    - *ld --verbose*
  - The most important sections are *.text*, *.data*, *.bss*, *.common*, *.vector*, *and .comment*
  - '.' is the VMA location counter, which always refer to the current location in a output object
  - '*' is a wildcard to match the filenames of all input objects

Makefile

```
CROSS   = /opt/iwmmxt-1.0.0/bin/arm-linux-
CC              = $(CROSS)gcc
OBJCOPY         = $(CROSS)objcopy
CFLAGS = -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -mcpu=xscale -mtune=xscale
export CC OBJCOPY CFLAGS

HOSTCC          = gcc
export HOSTCC

TOPDIR          := $(shell if test -n $$PWD; then echo $$PWD; else pwd; fi)
HPATH           := $(TOPDIR)/include
export TOPDIR HPATH

BOOTADDR = $(shell utils/bootaddr)              # shell script
export BOOTADDR                                 # define BOOTADDR       0xA3F00000          include/board.h

include config                                  # myipaddr=192.168.100.50     destipaddr=192.168.100.100


#.SILENT :
all : setup
                make -C src

clean :
                make clean -C src
                make clean -C utils

proper :
                make proper -C src
```

boot loader / monitor

```
dep :
        make dep -C src

setup : utils/inetaddr config
        perl -pi -e "s/(\.myipaddr   =)[^\n]*/\$\1 `utils/inetaddr $(myipaddr)`,/" src/setup.c
        perl -pi -e "s/(\.destipaddr =)[^\n]*/\$\1 `utils/inetaddr $(destipaddr)`,/" src/setup.c
        perl -pi -e "s/(\.myhaddr    =)[^\n]*/\$\1 `utils/macaddr`,/" src/setup.c
        perl -pi -e "s/(\.sid        =)[^\n]*/\$\1 `utils/setupid`,/" src/setup.c


utils/inetaddr :
        make -C utils
```

- **Note:**
  - ***test –n STRING*** **: the length of STRING is nonzero**
  - ***test –x FILE*** **: FILE exists and is executable**
  - ***patsubst PATTERN, REPLACEMENT, TEXT***
    - Finds whitespace-separated words in text that match pattern and replaces them with replacement.
    - pattern may contain a `%' which acts as a wildcard, matching any number of any characters within a word.

Immediate / Deferred

- **Immediate / Deferred**
  - *immediate* := *immediate*
  - *immediate* = *deferred*
- **GNU make does its work in two distinct phases.**
  - During the first phase it reads all the makefiles, included makefiles, etc. and internalizes all the variables and their values, implicit and explicit rules, and constructs a dependency graph of all the targets and their prerequisites.
  - During the second phase, make uses these internal structures to determine what targets will need to be rebuilt and to invoke the rules necessary to do so.
  - It's important to understand this two-phase approach because it has a direct impact on how variable and function expansion happens; this is often a source of some confusion when writing makefiles.

Immediate / Deferred

- **immediate**
  - » **if it happens during the first phase: in this case make will expand any variables or functions in that section of a construct as the makefile is parsed.**
- **deferred**
  - » **if expansion is not performed immediately. Expansion of deferred construct is not performed until either the construct appears later in an immediate context, or until the second phase.**

Makefile : capture

setup.c

```
static struct setup_t __bsetup _setup = {
                .sid       = 0x13cc66eb,
                .size      = (uint32)&loader_size,
                .myipaddr  = 0x3264a8c0,
                .destipaddr = 0x6464a8c0,
                .myhaddr   = {0x00,0x08,0xc9,0x62,0x5a,0xeb},
                .autoboot  = "load kernel; boot",
};
```

```
make -C utils
make[1]: Entering directory `/pxa270/bootloader/bboot-1.0.1/utils'
gcc -Wall -O6 -s -o macaddr macaddr.c
gcc -Wall -O6 -s -o setupid setupid.c
gcc -Wall -O6 -s -o inetaddr inetaddr.c
make[1]: Leaving directory `/pxa270/bootloader/bboot-1.0.1/utils'
perl -pi -e "s/(\.myipaddr  =)[^\n]*/\1 `utils/inetaddr 192.168.100.50`,/" src/setup.c
perl -pi -e "s/(\.destipaddr =)[^\n]*/\1 `utils/inetaddr 192.168.100.100`,/" src/setup.c
perl -pi -e "s/(\.myhaddr   =)[^\n]*/\1 `utils/macaddr`,/" src/setup.c  # random number
perl -pi -e "s/(\.sid       =)[^\n]*/\1 `utils/setupid`,/" src/setup.c        # random number
make -C src
make[1]: Entering directory `/pxa270/bootloader/bboot-1.0.1/src'
echo "compile start.S"
compile start.S
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o start.o -c start.S -I/pxa270/bootloader/bboot-1.0.1/include -D__ASSEMBLY__ -include config.h
echo "compile memsetup.S"
compile memsetup.S
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o memsetup.o -c memsetup.S -I/pxa270/bootloader/bboot-1.0.1/include -D__ASSEMBLY__ -include config.h
```

boot loader / monitor

```
echo "compile setup.c"
compile setup.c
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o setup.o -c setup.c -I/pxa270/bootloader/bboot-1.0.1/include -include config.h -include types.h
echo "compile memcpy.S"
compile memcpy.S
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o memcpy.o -c memcpy.S -I/pxa270/bootloader/bboot-1.0.1/include -D__ASSEMBLY__ -include config.h
.............
.............

echo "compile main.c"
compile main.c
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o main.o -c main.c -I/pxa270/bootloader/bboot-1.0.1/include -include config.h -include types.h
echo "compile time.c“
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o linux.o -c linux.c -I/pxa270/bootloader/bboot-1.0.1/include -include config.h -include types.h
.............
.............
echo "compile pxafb.c"
compile pxafb.c
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -Wall -O2 -nostdinc -fomit-frame-pointer -fPIC -fno-strict-aliasing -mstructure-size-boundary=8 -
mcpu=xscale -mtune=xscale -o pxafb.o -c pxafb.c -I/pxa270/bootloader/bboot-1.0.1/include -include config.h -include types.h
sed -e "s/BOOTADDR/0xa3f00000/" boot.lds.in > boot.lds
echo "create boot"
create boot
/opt/iwmmxt-1.0.0/bin/arm-linux-gcc -static -nostdlib -T boot.lds -o boot.elf32 start.o memsetup.o setup.o memcpy.o fixgpio.o partition.o main.o
time.o string.o command.o stdio.o vsprintf.o ctype.o cmddebug.o gpio.o ieb.o network.o bootp.o tftp.o cs8900.o flash.o serial.o xmodem.o
linux.o pxafb.o /opt/iwmmxt-1.0.0/lib/gcc/arm-linux/3.4.3/libgcc.a
/opt/iwmmxt-1.0.0/bin/arm-linux-objcopy -O binary -R .bss -R .note -R .comment -S boot.elf32 /pxa270/bootloader/bboot-1.0.1/boot
make[1]: Leaving directory `/pxa270/bootloader/bboot-1.0.1/src'
```

linker script : boot.lds

```
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS {
        . = 0xa3f00000;    BOOTADDR
        __boot_start = .;
        .text                   ALIGN(4) : {    *(.text)          }

        .setup                  ALIGN(4) : {    setup_block = .;
                                                *(.setup)
                                                setup_block_end = .; }

        .rodata                 ALIGN(4) : {    *(.rodata)        }

        .data                   ALIGN(4) : {    *(.data)          }

        .got                    ALIGN(4) : {    *(.got)           }

        __boot_end = .;

        .bss                    ALIGN(16) : {   bss_start = .;
                                                *(.bss)
                                                *(COMMON)
                                                bss_end = .;    }

        .comment                ALIGN(16) : {   *(.comment)     }

        stack_point = __boot_start + 0x00100000;
        loader_size = __boot_end - __boot_start;
        setup_size  = setup_block_end - setup_block;
}
```

boot loader / monitor

- **make –C utils**

```
CC=gcc
CFLAGS=-Wall -O6 -s

all :
        $(CC) $(CFLAGS) -o macaddr macaddr.c
        $(CC) $(CFLAGS) -o setupid setupid.c
        $(CC) $(CFLAGS) -o inetaddr inetaddr.c

clean :
        rm -f macaddr setupid inetaddr
}
```

- **make –C src**

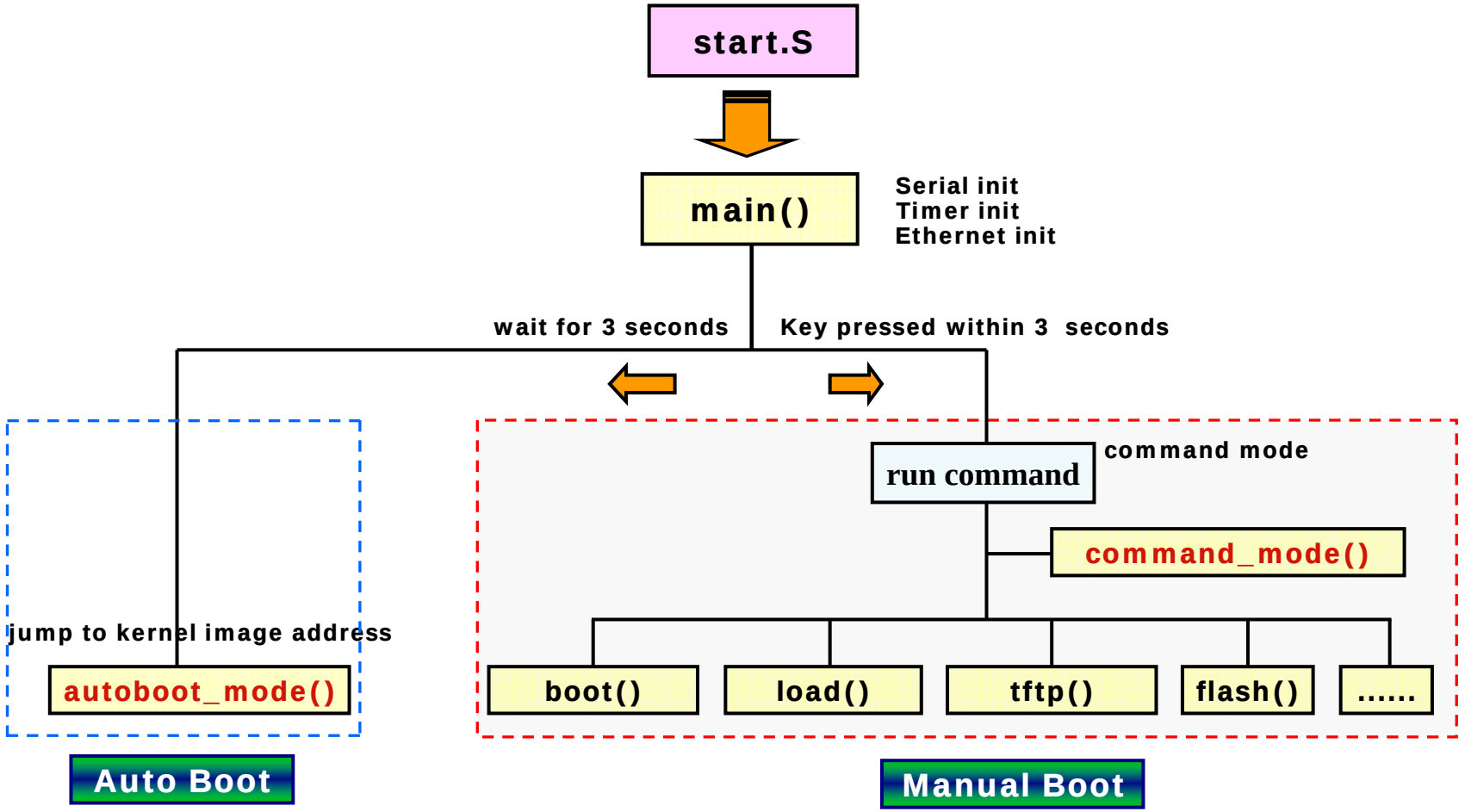
```
objfiles  = start.o memsetup.o setup.o memcpy.o fixgpio.o partition.o
objfiles += main.o time.o string.o command.o stdio.o vsprintf.o ctype.o cmddebug.o gpio.o ieb.o
objfiles += network.o bootp.o tftp.o cs8900.o
objfiles += flash.o serial.o xmodem.o linux.o pxafb.o

libgcc = $(shell $(CC) --print-libgcc-file-name)

binfile = boot

all : $(objfiles) boot.lds
        echo "create $(binfile)"
        $(CC) -static -nostdlib -T boot.lds -o $(binfile).elf32 $(objfiles) $(libgcc)
        $(OBJCOPY) -O binary -R .bss -R .note -R .comment -S $(binfile).elf32 $(TOPDIR)/$(binfile)
```

make –C src : cont.

```
%.o : %.S
		echo "compile $<"
		$(CC) $(CFLAGS) -o $@ -c $< $(patsubst %,-I%, $(HPATH)) -D__ASSEMBLY__ -include config.h

%.o : %.c
		echo "compile $<"
		$(CC) $(CFLAGS) -o $@ -c $< $(patsubst %,-I%, $(HPATH)) -include config.h -include types.h


dep : .depend

.depend : $(wildcard *.S) $(wildcard *.c)
		rm -f .depend; touch .depend
		for x in $(wildcard *.S); do $(CC) $(CFLAGS) -M -MT $${x/%.S/.o} $$x -I$(HPATH) >> .depend; done
		for x in $(wildcard *.c); do $(CC) $(CFLAGS) -M -MT $${x/%.c/.o} $$x -I$(HPATH) >> .depend; done

ifeq (.depend,$(wildcard .depend))
include .depend
endif

clean :
		rm -f *.o boot.lds *.elf32 $(TOPDIR)/$(binfile)


proper : clean
		rm -f .depend


boot.lds : boot.lds.in $(HPATH)/config.h
		sed -e "s/BOOTADDR/$(BOOTADDR)/" boot.lds.in > boot.lds
```

Boot Loader / Monitor

- **boot : Boot loader / monitor**



- **start.S**

```
#include <config.h>
#include <hardware.h>

.section .text
.global _start
_start :
        bl      define_gpio         @ fixgpio.S
        bl      clock_enable
        bl      setup_memory        @ memsetup.S setup static and dynamic memory

        @ copy bootloader to dynamic memory area
        ldr     r0, =0x00
        ldr     r1, =__boot_start   @ boot.lds
        ldr     r2, =__boot_end
1:      ldmia   r0!, {r3-r10}
        stmia   r1!, {r3-r10}
        cmp     r1, r2
        blt     1b

        @ clear bss area
        mov     r3, #0x00
        mov     r4, #0x00
        mov     r5, #0x00
        mov     r6, #0x00
        ldr     r0, =bss_start
        ldr     r1, =bss_end
1:      stmia   r0!, {r3-r6}
        cmp     r0, r1
        blt     1b
```

ldr macro

start.S - cont.

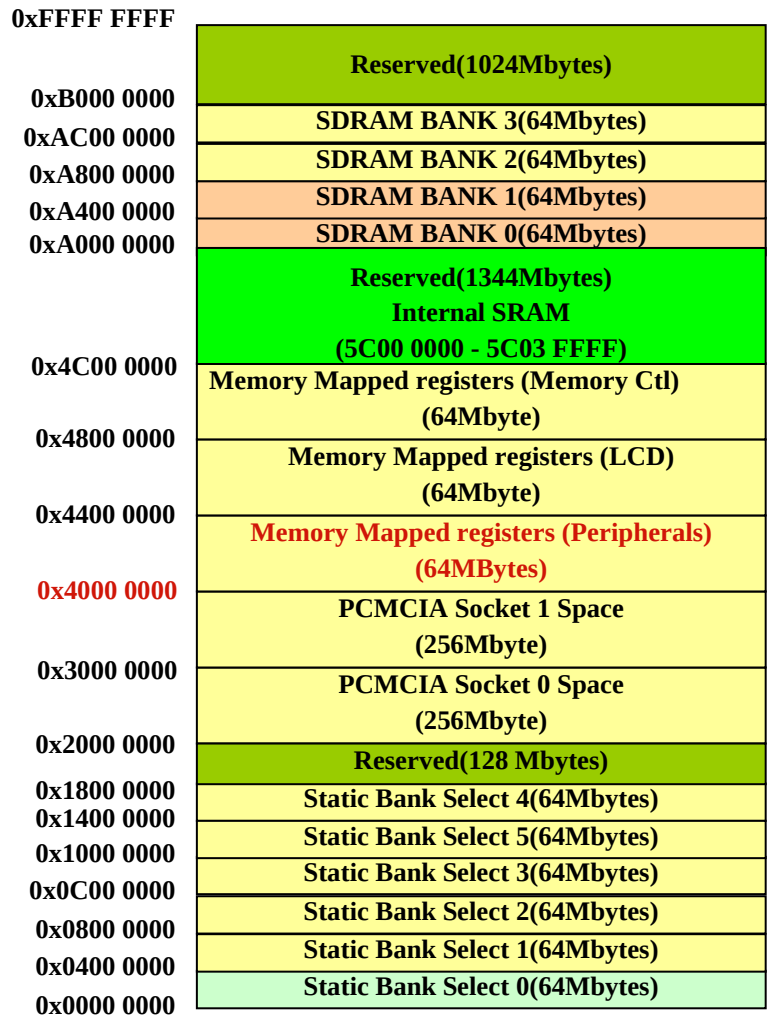
```
                @ set stack point
                ldr             sp, =stack_point-4

                @ jump to c code
                ldr             pc, =main

clock_enable :
#if defined(CONFIG_PXA25x)
                ldr             r0, =0x0001FFFF
#elif defined(CONFIG_PXA27x)
                ldr             r0, =0x01FFFFFF
#endif
                ldr             r1, =CKEN
                str             r0, [r1]
                mov             pc, lr
```

include/pxareg.h

```
/* Core Clock  */

#define CCCR            __REG(0x41300000)  /* Core Clock Configuration Register */
#define CKEN            __REG(0x41300004)  /* Clock Enable Register */
#define OSCC            __REG(0x41300008)  /* Oscillator Configuration Register */
#define CCSR            __REG(0x4130000C)  /* Core Clock Status Register */
```

Memory Map and Registers

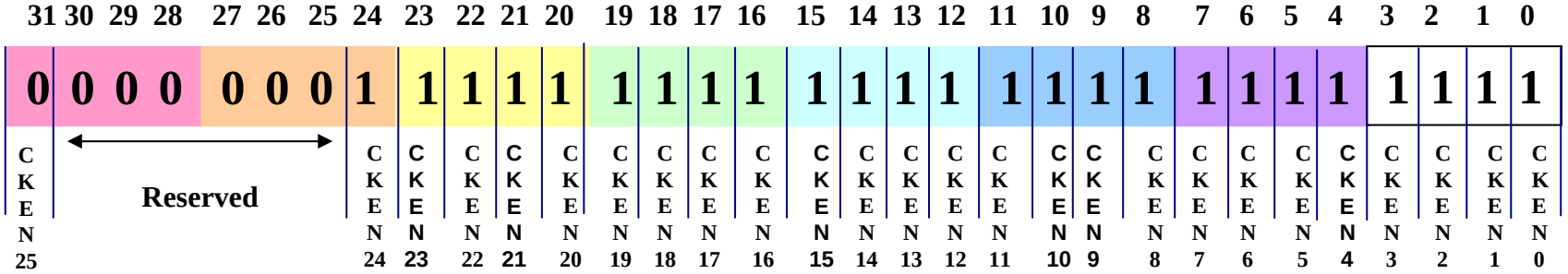
» Intel® PXA27x Processor Family Developer's Manual



Peripheral Module Address Summary

Unit	Address
DMA Controller	0x4000_0000
UART1—Full Function UART	0x4010_0000
UART2—Bluetooth UART	0x4020_0000
Standard I²C Bus Interface Unit	0x4030_0000
I²S Controller	0x4040_0000
AC '97 Controller	0x4050_0000
USB Client Controller	0x4060_0000
UART 3—Standard UART	0x4070_0000
Fast Infrared Communications Port	0x4080_0000
RTC	0x4090_0000
OS Timers	0x40A0_0000
PWM 0 and 2	0x40B0_0000
PWM 1 and 3	0x40C0_0000
Interrupt Controller	0x40D0_0000
GPIO Controller	0x40E0_0000
Power Manager	0x40F0_0000
Reset Controller	0x40F0_0000
Power Manager I²C	0x40F0_0180
Synchronous Serial Port 1	0x4100_0000
MultiMediaCard/SD/SDIO Controller	0x4110_0000
reserved	0x4120_0000
Clocks Manager	0x4130_0000
Mobile Scalable Link (MSL)	0x4140_0000
Keypad Interface	0x4150_0000
Universal Subscriber ID (USIM) Interface	0x4160_0000
Synchronous Serial Port2	0x4170_0000
Memory Stick Host Controller	0x4180_0000
Synchronous Serial Port3	0x4190_0000

Clock Enable Register (CKEN)



Name	Description	Name	Description
CKEN[25]	TPM Unit Clock Enable	CKEN[11]	USB Client Unit Clock Enable
CKEN[24]	Quick Capture Interface Clock Enable		
CKEN[23]	SSP1 Unit Clock Enable	CKEN[10]	USB Host Unit Clock Enable
CKEN[22]	Memory Controller	CKEN[9]	OS Timer Unit Clock Enable
CKEN[21]	Memory Stick Host Controller	CKEN[8]	I²S Unit Clock Enable
CKEN[20]	Internal Memory Clock Enable	CKEN[7]	BTUART Unit Clock Enable
CKEN[19]	Keypad Interface Clock Enable	CKEN[6]	FFUART Unit Clock Enable
CKEN[18]	USIM Unit Clock Enable	CKEN[5]	STUART Unit Clock Enable
CKEN[17]	MSL interface Unit Clock Enable	CKEN[4]	SSP3 Unit Clock Enable
CKEN[16]	LCK Controller Clock Enable	CKEN[3]	SSP2 Unit Clock Enable
CKEN[15]	Power Manager I²C Unit Clock Enable	CKEN[2]	AC '97 Controller Clock Enable
CKEN[14]	I²C Unit Clock Enable	CKEN[1:0]	0b00 = All PWMs disabled 0b01 = All PWMs enabled 0b10 = All PWMs enabled 0b11 = All PWMs enabled
CKEN[13]	Infrared Port Clock Enable		
CKEN[12]	MMC Controller Clock Enable		

* All Unit Clock Enable Bit;

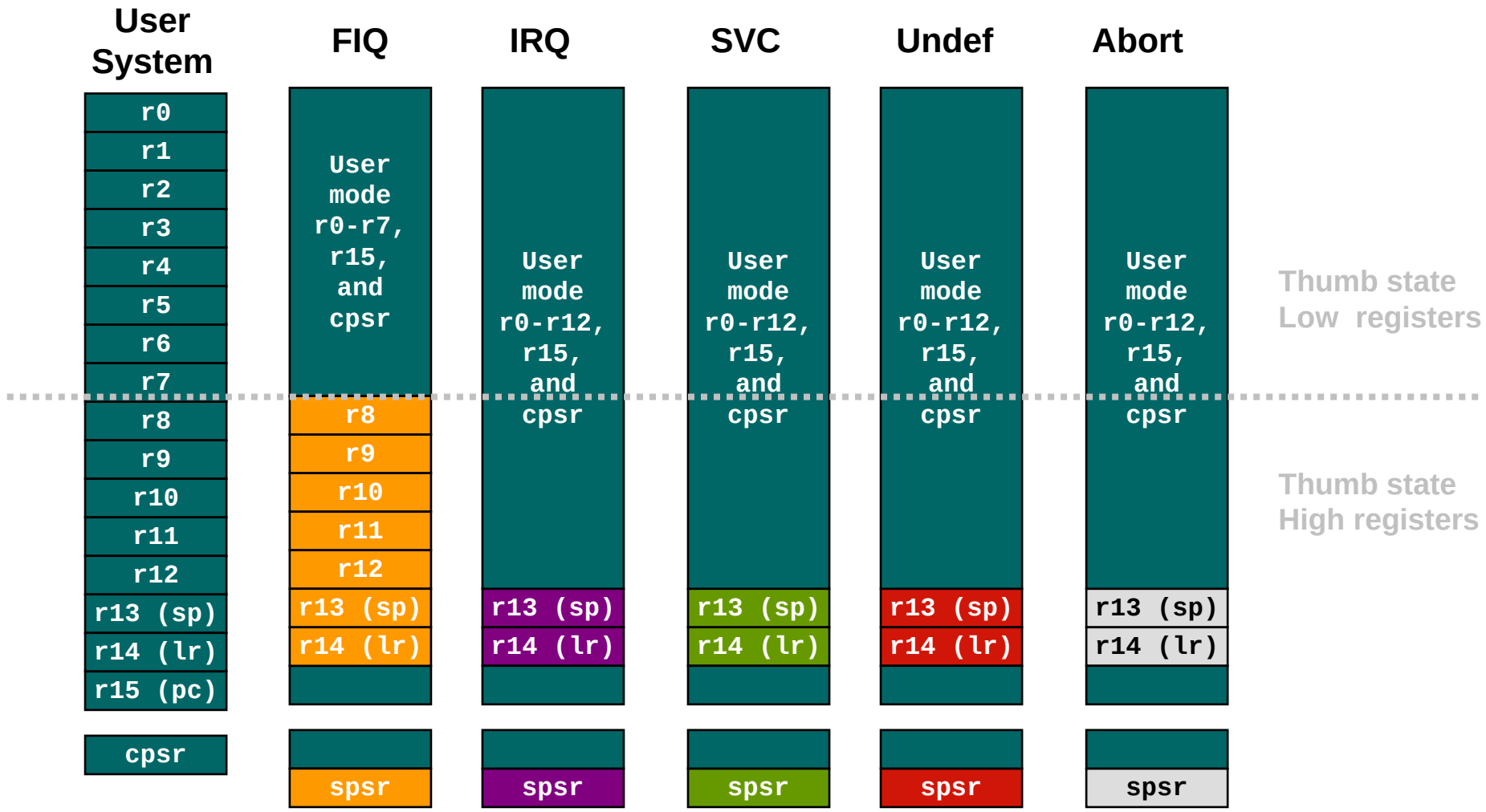
0 – Clock disable
1 – Clock enable

0x4130_0000	CCCR	Core Clock Configuration register
0x4130_0004	CKEN	Clock Enable register
0x4130_0008	OSCC	Oscillator Configuration register
0x4130_000C	CCSR	Core Clock Status register
0x4130_0010-0x413F_FFFC	—	reserved

ARM Processor Modes

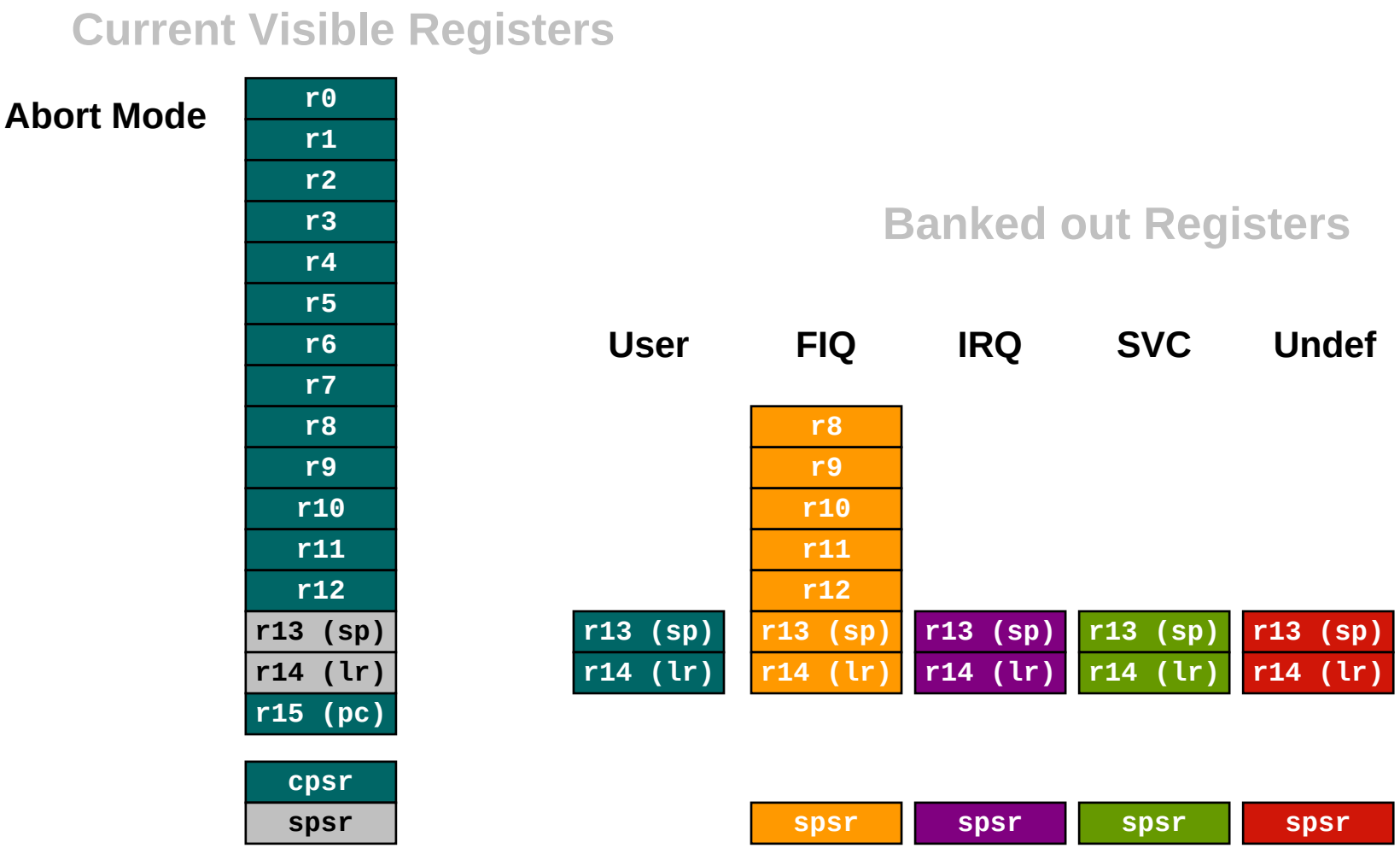
- **Privileged / Non-privileged**
  - Privileged : full read & write access to the cpsr
  - Non-privileged : only read access to the control field / read & write access to the condition flags in the cpsr
- **Seven operating modes** = 6 Privileged + 1 Non-privileged
  - User : unprivileged mode (most tasks run for programs and applications)
  - FIQ : high priority (fast) interrupt is raised
  - IRQ : low priority (normal) interrupt is raised
  - Supervisor : entered on reset and OS kernel operates in (Software Interrupt instruction is executed)
  - Abort : failed memory access (memory access violations)
  - Undef : used to handle undefined instructions
  - System : special version of user mode, full read & write access to the cpsr (privileged mode using the same registers as user mode)

Register Set



Note: System mode uses the User mode register set

Register Set

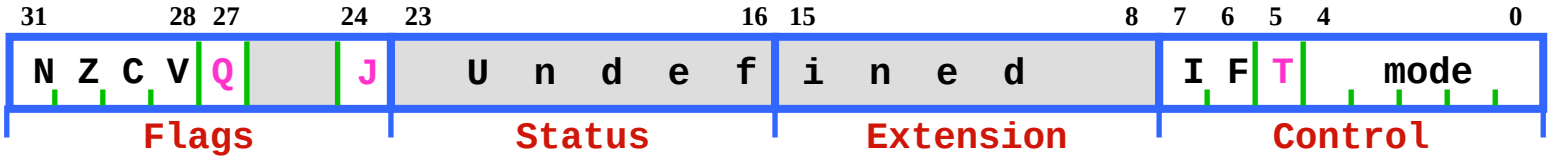


Registers

- **37 Registers : 32-bits long**
  - 1 dedicated program counter
  - 1 dedicated current program status register
  - 5 dedicated saved program status registers
  - 30 general purpose registers

- **Accessible registers**
  - a particular set of r0-r12 registers
  - a particular r13 (the stack pointer, sp) and r14 (the link register, lr)
  - the program counter, r15 (pc)
  - the current program status register (cpsr)
  - a particular spsr (saved program status register)

cpsr



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Architecture 5TE/J only
  - Saturation bit
- **J bit**
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- **Interrupt Disable bits.**
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.
- **T Bit**
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **Mode bits**
  - Specify the processor mode

flags

- **Condition codes**

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

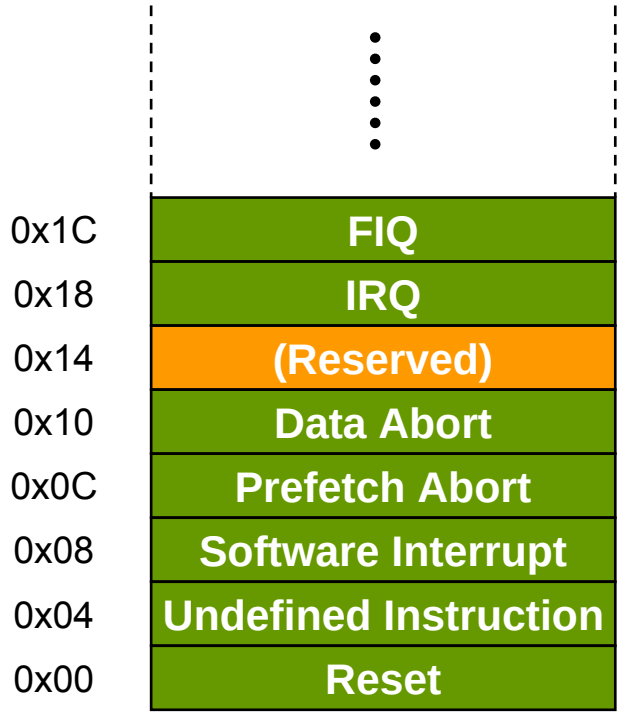
cpsr

- **Mode bits**

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	Yes	10111
FIQ	fiq	Yes	10001
IRQ	irq	Yes	10010
Supervisor	svc	Yes	10011
System	sys	Yes	11111
Undefined	und	Yes	11011
User	usr	No	10000

- **When an exception occurs, the ARM:**
  - **Copies CPSR into SPSR**
  - **Sets appropriate CPSR bits**
    - Change to ARM state
    - Change to exception mode
    - Disable interrupts (if appropriate)
  - **Stores the return address in LR**
  - **Sets PC to vector address**
- **To return, exception handler needs to:**
  - **Restore CPSR from SPSR**
  - **Restore PC from LR**

This can only be done in ARM state.

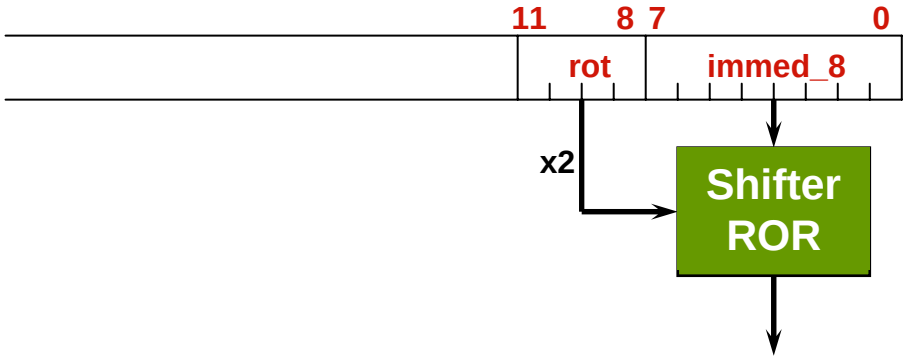


Vector Table

Vector table can be at **`0xFFFF0000`** on ARM720T and on ARM9/10 family devices

Immediate constants

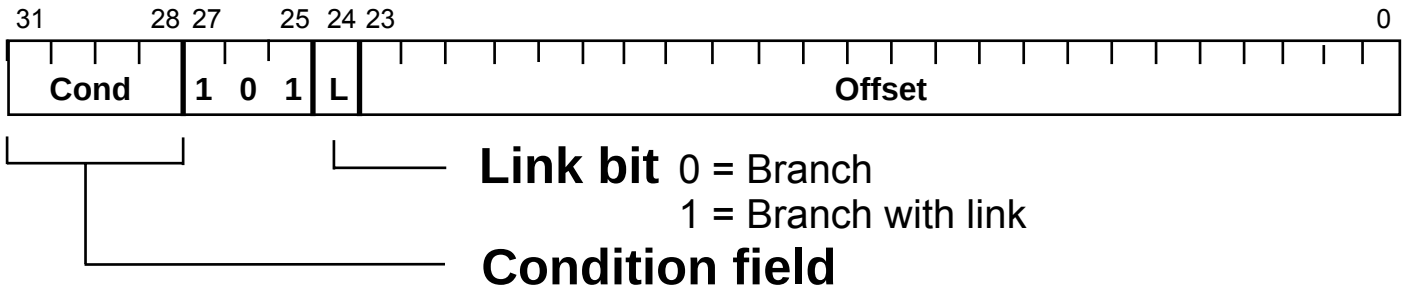
- **No ARM instruction can contain a 32 bit immediate constant**
  - » **All ARM instructions are fixed as 32 bits long**
- **The data processing instruction format has 12 bits available for operand2**



- **To allow larger (32 bit) constants to be loaded, the assembler offers a pseudo-instruction (macro):**
  - » `LDR rd, =const`

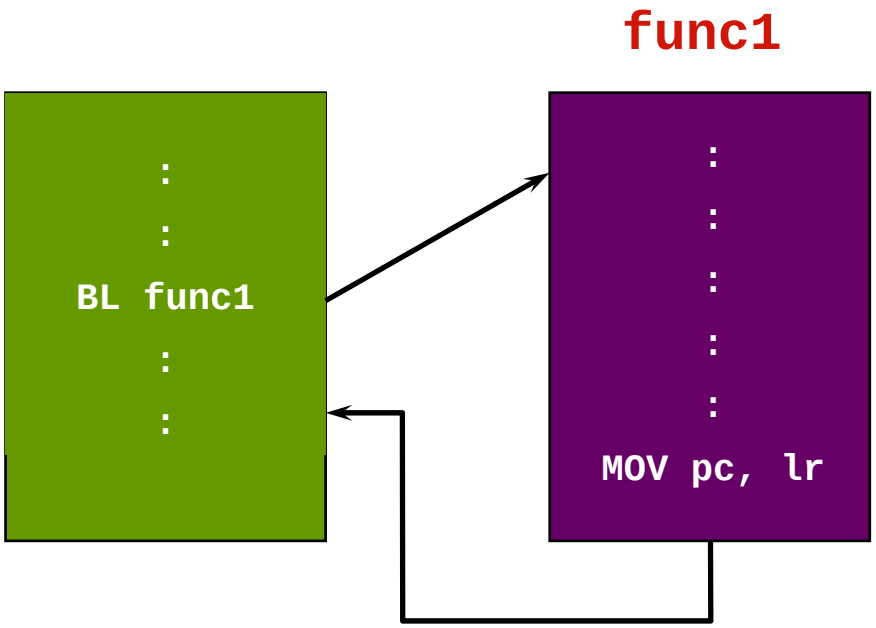
Branch instructions

- **Branch :**
  - **`B{<cond>} label`**
    - `pc = label`
- **Branch with Link :**
  - **`BL{<cond>}subroutine_label`**
    - `pc = subroutine_label`
    - `lr = address of the next instruction after the BL`



Branches and Subroutines

- **B <label>**
  - **PC relative. ±32 Mbyte range.**
- **BL <subroutine>**
  - **Stores return address in LR**
    - lr = address of the next instruction after the BL
  - **Returning implemented by restoring the PC from LR**



Load/Store Multiple Instruction

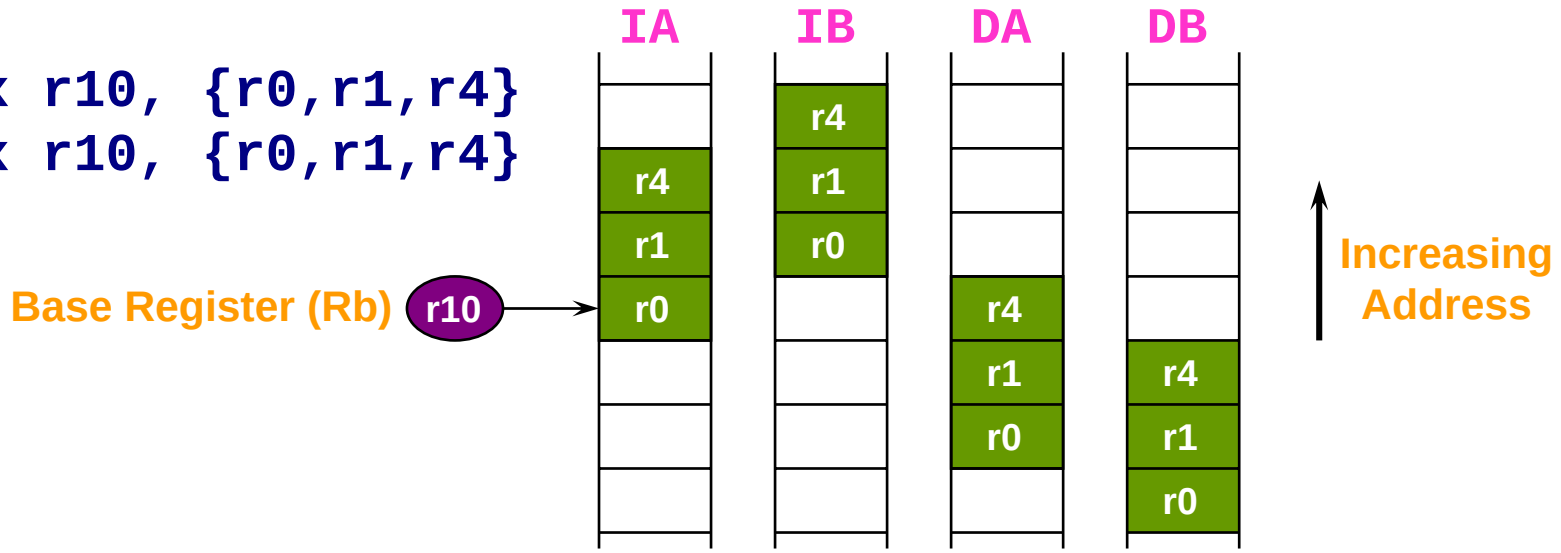
- **Syntax:**

  `<LDM|STM>` {<cond>} Rb{!}, <register list>

- **4 addressing modes:**

LDMIA / STMIA	increment after
LDMIB / STMIB	increment before
LDMDA / STMDA	decrement after
LDMDB / STMDB	decrement before

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



LDMIA Instruction

- **LDMIA r0!, {r1-r3}**

Pre-condition

Memory

	Address	Data	
	0x80020	0x000000005	
	0x8001c	0x000000004	
	0x80018	0x000000003	r3=0x00000000
	0x80014	0x000000002	r2=0x00000000
r0=0x80010	0x80010	0x000000001	r1=0x00000000
	0x8000c	0x000000000	

Post-condition

Memory

	Address	Data	
	0x80020	0x000000005	
r0=0x8001c	0x8001c	0x000000004	
	0x80018	0x000000003	r3=0x00000003
	0x80014	0x000000002	r2=0x00000002
	0x80010	0x000000001	r1=0x00000001
	0x8000c	0x000000000	

* Note : ! indicates that the instruction writes the calculated address back to the base address register.

main.c

```
int main(void){
        int count;
        bool autoboot=true;

        uart_init();
        uart_init();
        pxafb_disable();
        time_init();
        config_init();

        printf("\033[H\033[J\n");                       // clear screen.
        printf(" %s : bootloader for xhyper board\n", PACKAGE);
        printf(" Copyright (C) 2002-2004 Hybus Co,. ltd.\n");
        printf(" support : http://www.hybus.net\n");

        iflash_init();
        eth_init();
        printf(" IEB board TEST\n");
        ieb_init();                                     //270_tku ieb init
        printf(" autoboot in progress, press any key to stop.");

        count = 2000;
        autoboot = true;
        do {
                if (getc() != -1){ autoboot=false; break; }
                if (!(--count % 1000)) putc('.');
                mdelay(1);
        } while (count);
        putc('\n');
```

main.c

- **main.c – cont.**

```
        if (autoboot){
                printf(" autoboot started.\n");
                autoboot_mode();
        }

        printf(" autoboot aborted\n");
        printf(" type \"help\" to get a list of commands\n");
        command_mode();
        while (1);
        return 0;
}
```

- **command.c**
  - » **autoboot_mode() → command parsing → do_boot()**

do_boot()

- **do_boot()**

```
static bool do_boot(int argc, char **argv){
        void (*kernel)(int zero, int arch);

        if (argc == 1){
                struct map *mp;
                mp = find_map("kernel");
                if (!mp){ printf(" can't found map for kernel.\n"); goto invalid; }
                kernel = (void *)mp->dramb;
        } else if (argc == 2){
                bool res;
                ulong tmp;

                res = strtoul(argv[1], &tmp, 16);          # kernel start address = &tmp
                if (!res) goto invalid;
                kernel = (void *)tmp;
        } else goto invalid;
        if (!get_kernel_size(kernel)){ printf(" error: kernel is not exists.\n"); return false; }
        create_tags();
        printf("starting kernel ...\n");
        kernel(0, 200);
        return true;
invalid :
        boot_usage();
        return false;
}
```

Kernel Compile

- **zImage : kernel**
  - ***cd /pxa270/kernel***
  - ***tar xzvf linux-2.6.11-h270-tku_v1.1.tar.gz***
  - ***cd linux-2.6.11-h270-tku_v1.1***
  - ***make xhyper270tku_defconfig***
    - /linux-2.6.11-h270-tku_v1.1/arch/arm/configs/xhyper270tku_defconfig → /linux-2.6.11-h270-tku_v1.1/.config
  - ***make menuconfig***
    - *make config* : text based interactive
    - *make menuconfig* : text based menu driven
      - → kernel/linux-2.6.11-h270-tku_v1.1/scripts/kconfig/Makefile
    - *make xconfig* : Xwindows-GUI – *cd /usr/src/linux-2.4*
  - **Kernel configuration : Manual p.155 – p. 218**
  - ***make zImage***
  - ***cp arch/arm/boot/zImage /tftpboot***
  - ***tftp zImage kernel : at boot monitor***
  - ***flash kernel***

Root File system

- **rootfs.img : Root File System**
  - **JFFS2 : Journaling Flash File System**
  - **Kernel configuration : Manual p.213 – p. 218**
  - **Flash Memory Definitions**
    - kernel/linux-2.6.11-h270-tku_v1.1/drivers/mtd/maps/xhyper270tku.c
  - ***cd /pxa270/filesystem***
  - ***cd rootfs , ls , & cd ..***
  - ***rm –rf rootfs.img***
  - ***./mkjffs2 : script → mkfs.jffs2***
  - ***cp rootfs.img /tftpboot***
  - ***tftp rootfs.img root***

RAM Disk

- **RAM Disk**
  - **Using the RAM as a disk**
    - stored only temporarily
    - save and retrieve files very quickly
    - conserve power
  - ***dd if=/dev/zero of=ramdisk_img bs=1k count=8192***
  - ***file ramdisk_img***
  - ***mke2fs ramdisk_img***
  - ***mkdir ramdisk***
  - ***mount –o loop ./ramdisk_img ./ramdisk***
  - ***df***
  - ***cd ramdisk***
  - ***ls –al***
- **“loop” device**
  - **a device driver that allows an image file to be mounted as though it were a normal block device.**