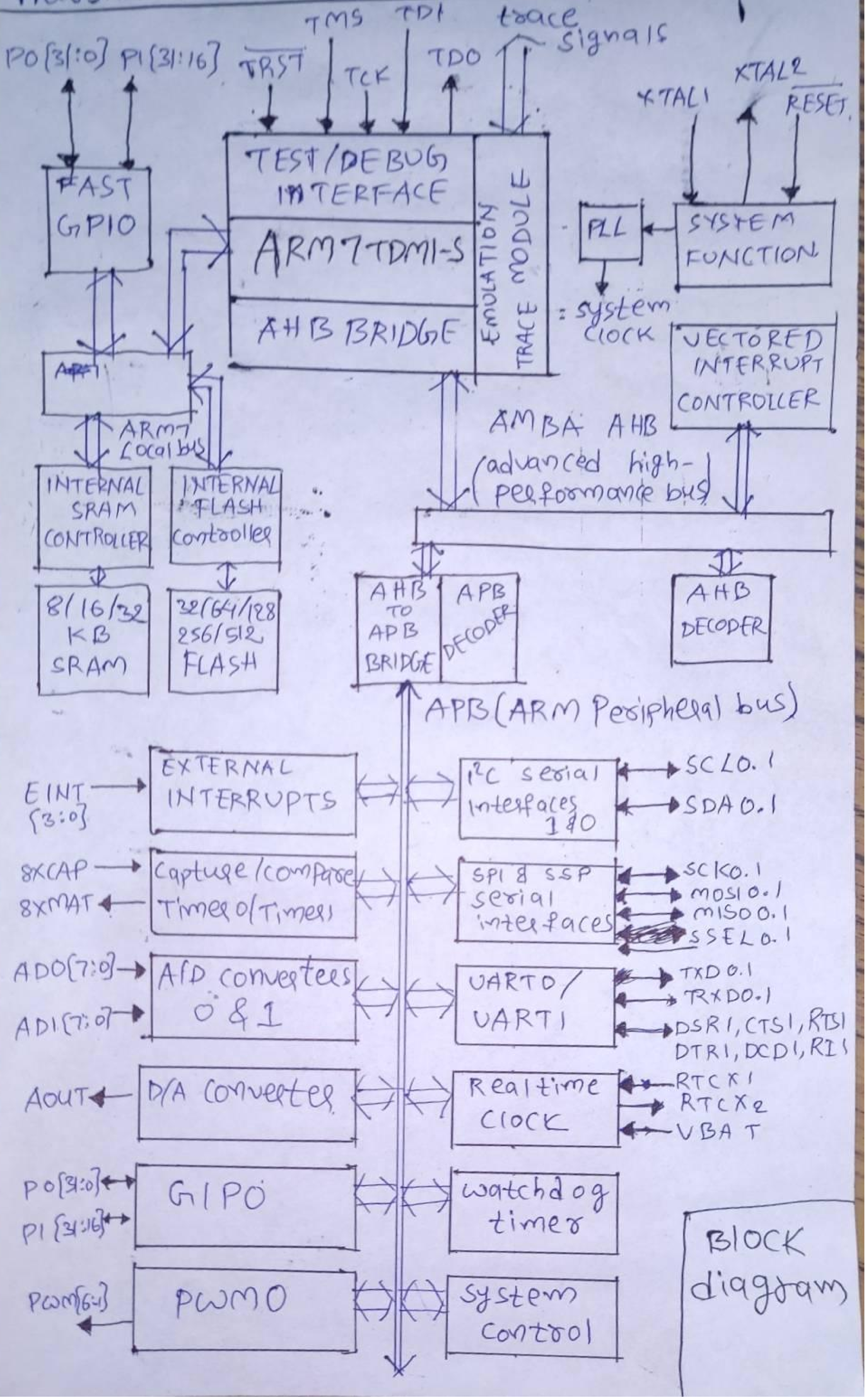
Assignment-2

Harshavardhan Alimi - 18EC10021



BLOCK diagram

* Key features of LPC213X ARM7 microcontroller:-

→ 16/32-bit ARM.7T.DMI-S microcontroller in a tiny LQFP64 or HVQFN64 package.

→ 8/16/32 KB of on-chip static RAM and 32/64/128/256/512 KB of on-chip flash program memory. 128-bit wide interface/accelerator enables high-speed of 60 MHz operation.

→ In-system programming / In-Application Programming (ISP/IAP) via on-chip bootloader software. Single flash sector or full chip erase in 400 ms & programing of 256 B in 1ms.

→ EmbeddedICE RT & Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high-speed tracing of instruction execution.

→ one (LPC2131/32) or two (LPC2134/36/38) 10-bit ADC's provide a 8-channel total of upto 16 analog inputs, with conversion times as low as 2.44 μs per channel.

→ Single 10-bit DAC provides variable analog output (LPC 2132/34/36/38).

→ Two 32-bit timers/ external event counters (with four capture & four compare channels each), PWM unit (six outputs) & watchdog.

→ Low power Real-time clock with independent power & dedicated. 32 KHz clock input.

→ multiple serial interface's including two UART's (16C550), two Fast I²C-bus (400 kbit(s)), SPI & SSP with buffering & variable data length capabilities.

→ vectored interrupt controller with configurable priorities & vector addresses.

→ upto 47 5V tolerant general purpose I/O pins in tiny package

→ upto 9 edge or level sensitive external interrupt pins available.

→ 60 MHz max CPU clock available. from programmable on-chip PLL with settling time of 100 μS.

→ on-chip integrated oscillator operates with external crystal in range of 1MHz to 30 MHz & with external oscillator up to 50 MHz

→ power saving modes include idle & power-down.

Scanned by TapScanner

→ Individual enable/disable of peripheral functions as well as peripheral clock scaling down for additional power optimization.

→ Processor wake-up from power-down mode via external interrupt or BOD.

→ Single power supply chip with POR & BOD circuits:-

* CPU operating voltage range of 3.0V to 3.6 V (3.3V ± 10%) with 5V tolerant I/O pads.

2> MAM :- (memory accelarator module) :-

The memory accelarator module is the key to the high instruction execution rate of LPC 213X family. The MAM is present in the local bus & sits between the Flash memory and ARM7 CPU.

The MAM is a compromise between the complexity of a full cache & the simplicity of allowing the processor to directly access the Flash memory. Like cache, the MAM attempts to have next ARM instruction in its Local memory in time for CPU to execute user code which is intelleaved between the two banks, so during sequential code execution, the code fetched from one bank into the MAM is being executed, while the next 128 bits of instructions from the second bank is being perfected. The complexities of the MAM are transparent to the user & are configured by two registers, the timing register & the control register. There are some additional registers, to provide runtime

Scanned by TapScanner

information on the effectiveness on the MAM. The group of stastical register which can be used to measure MAM's performance, are based around two counters, which record the access mode to the Flash & the access mode to the MAM buffers.

APB:-

The APB decoder determines the relationship between processor clock and clock used by peripheral devices. This serves 2 purposes. First, is to provide peripherals with desired PCLK via APB bus so that they can operate at the speed closer to the processor. In order to achieve this, APB bus may be slowed down by 0.5-2.5 times of PCLK. Because, the APB bus must work properly at powerup. (and its timing can't be altered if it doesn't work since the APB divider control registers reside on the APB bus. The default setup at RESET for APB bus to RUN is at 0.25 of PCLK. The 2nd purpose of APB divider is to allow power saving when an application doesn't require any peripherals to run at full processor rate. Because the APB divider is connected to the PLL output, the PLL remaining actual (if it was running) during idle mode.

Watchdog timer:-

The purpose of the watchdog is to reset the microcontroller with a reasonable amount of time if it enters the erroneous state. When enabled, the watchdog will generate a system reset (interrupt) if the user program fails to feed the watchdog timer within a predetermined amount of time.

3) ARM7TDMI :- Processor has a total of 37 registers

→ 31 General purpose 32-bit registers.

→ 6 status registers.

These registers are not all accessible at the same time. The processor state & operating mode determines which registors are available.

The ARM state register-set offers 16 general registers & one or 2 status registers that are accessible at any one time. In privileged modes, mode-specific banked registers become available.

The Thumb state register set is a subset of the ARM-state set.

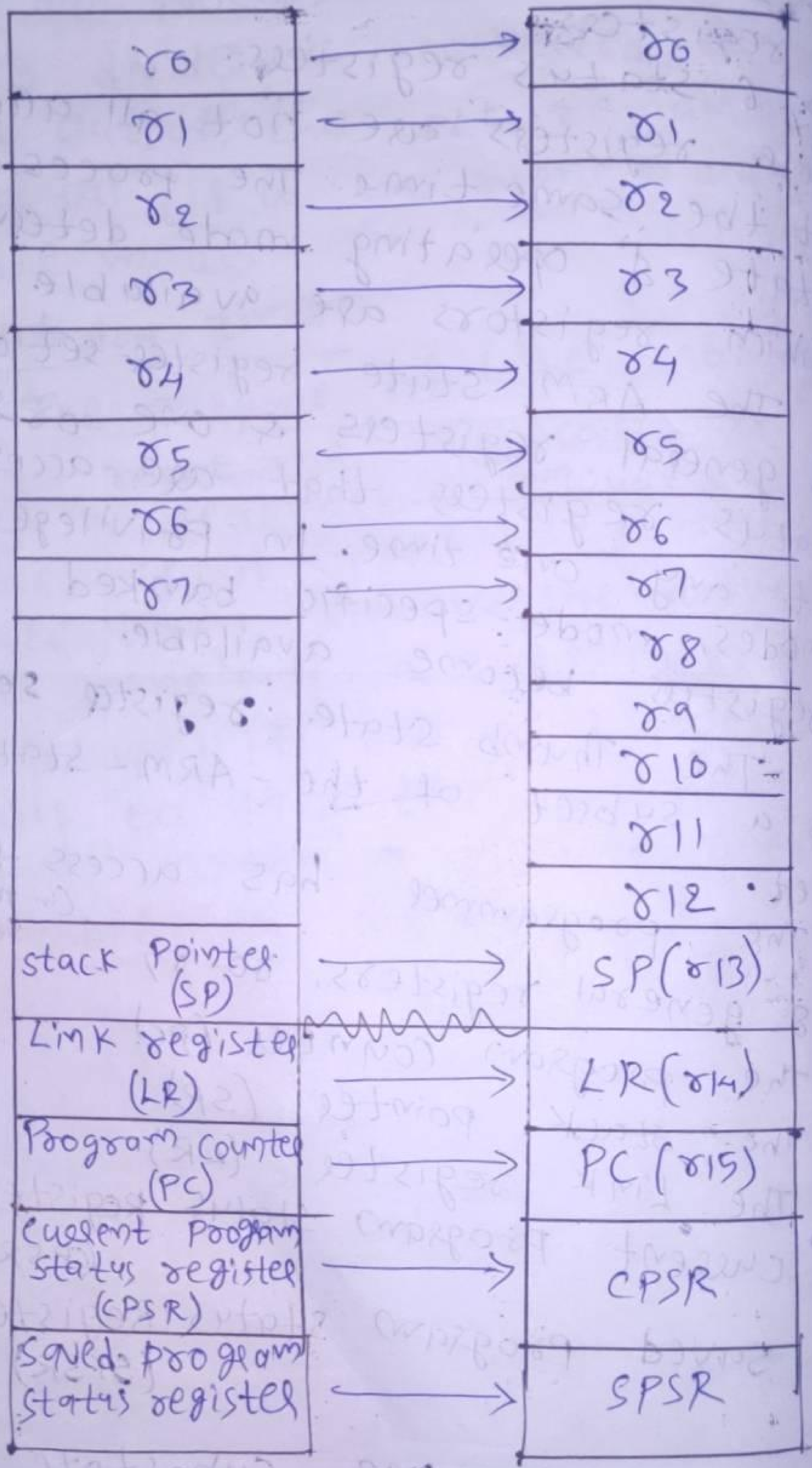
The programmer has access to:- (in Thumb state)

- 8 general registers, r0-r7
- The program counter (PC)
- The stack pointer (SP)
- The Link register (LR)
- current Program status Register (CPSR)
- saved Program status Register* (SPSR)

* - except in one sub-state.

Scanned by TapScanner

Mapping of ARM state & Thumb state registers



ARM-state general registers & program counter.

system & user	FIQ	supervisor	Abort	IRQ	undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	▲ r8-fiq	r8	r8	r8	r8
r9	▲ r9-fiq	r9	r9	r9	r9
r10	▲ r10-fiq	r10	r10	r10	r10
r11	▲ r11-fiq	r11	r11	r11	r11
r12	▲ r12-fiq	r12	r12	r12	r12
r13	▲ r13-fiq	▲ r13-svc	▲ r13-abt	▲ r13-irq	▲ r13-und
r14	▲ r14-fiq	▲ r14-svc	▲ r14-abt	▲ r14-irq	▲ r14-und
r15(PC)	r15(PC)	r15 (PC)	r15 (PC)	r15(PC)	r15(PC)

ARM-state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	▲ SPSR_fiq	▲ SPSR_svc	▲ SPSR_abt	▲ SPSR_irq	▲ SPSR_und

▲ = banked register.

Thumb-state general registers & Program counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
SP	▲ SP_fiq	▲ SP_svc	▲ SP_abt	▲ SP_irq	▲ SP_und
LR	▲ LR_fiq	▲ LR_svc	▲ LR_abt	▲ LR_irq	▲ LR_und
PC	PC	PC	PC	PC	PC

Thumb-state program registers.

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	▲ SPSR_fiq	▲ SPSR_svc	▲ SPSR_abt	▲ SPSR_irq	▲ SPSR_und

▲ = banked register

* processor contains a CPSR &
5 SPSR's for exception handlers
to use. The program status
registers:-
→ Hold information about
most recently performed
ALU operation.
→ control the enabling & disab
of interrupts
→ set processor operating mode

SPSR :- saved program status
register stores the current value
of ~~S~~CPSR when an exception is
taken so that the CPSR
can be restored after handing
exception.
CPSR:- continuous program status
register.

Scanned by TapScanner

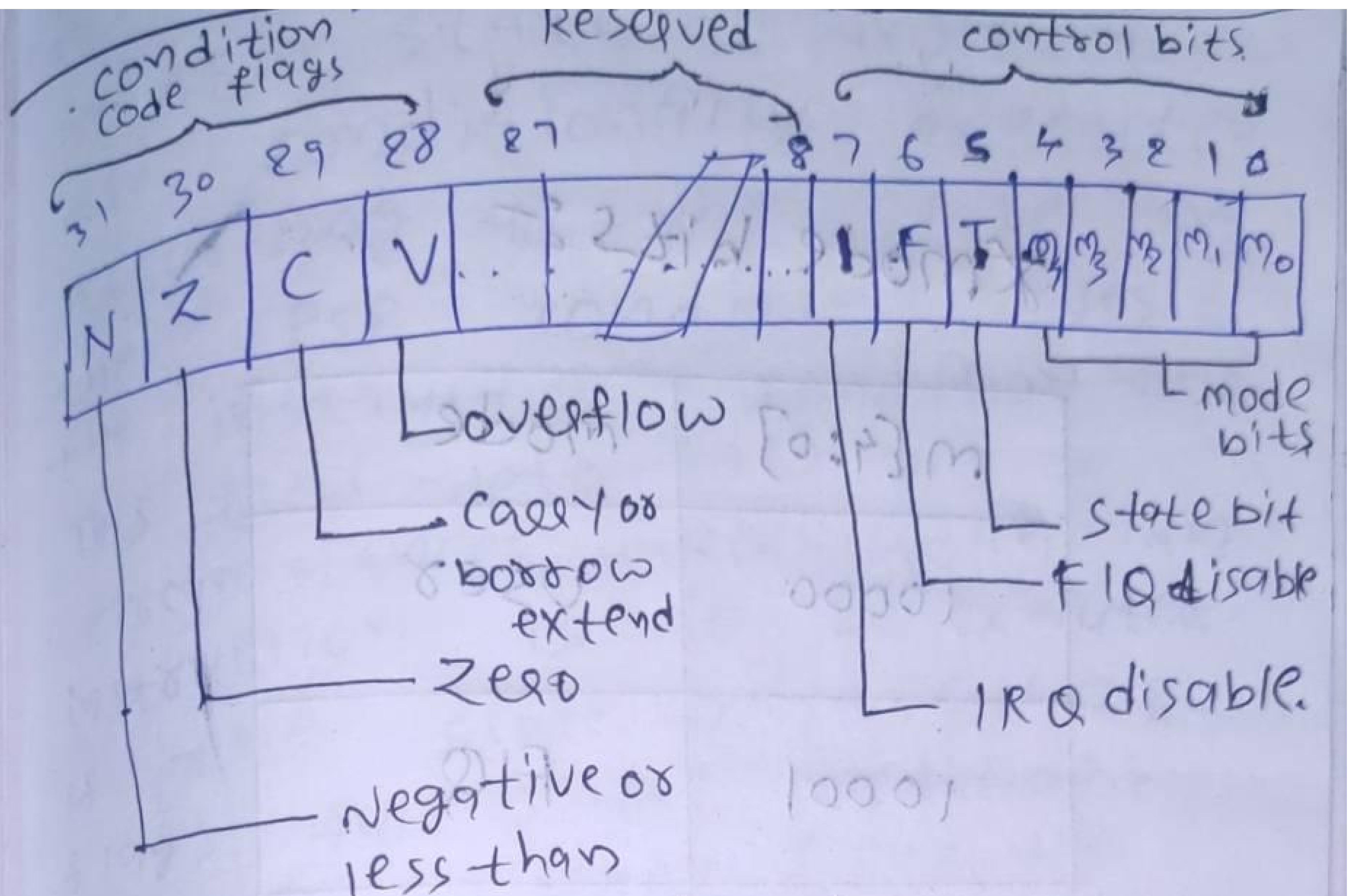


Fig. (CPSR & SPSR)
program status Register format

* Interrupt disable bits :- (7, 6)
The I&F bit are interrupt disable bits:-
• when I bit is set, IRQ interrupts are disabled.
• when F bit is set, FIQ interrupts are disabled.

* T-bit :- (5)
The T-bit reflects the operating state.
• when T-bit is set, the processor is executing in Thumb state.
• when T-bit is clear, the processor executing in ARM state.
The operating state is reflected on external signal TBIT.

*mode bits:-

m[4:0]	mode
10000	user
10001	FIQ
10010	IRQ
10011	supervisor
10111	Abort
11011	undefined
11111	system

* An illegal value programmed into m[4:0] causes the processor to enter an unrecoverable state. if this occurs, apply reset.

Scanned by TapScanner

4) There are several ways in which we can clock ARM micro controller.
* one way is to use an external clock with duty cycle of 50% & in a frequency range 1-50 MHz connected to XTAL1 pin.
* second way is by connecting external crystal oscillator but with lower frequency range of 1-30 MHz.
we can also use on-chip PLL oscillator, but the external clock frequency should not exceed the range of 25 MHz & fall below 10 MHz.

PLL mechanism (LPC 213X ARM7):-
⇒ PLL is used to generate system clock frequency between 10-25 MHz. PLL may multiply frequency to range from 10-60 MHz & 48 MHz for USB if used. PLL uses frequency multiplier which can be in a range from 1-32, In real world situation this value should not be greater than 6. due to upper limit. PLL generates allows running ARM at high speed with low frequency oscillator connected. Also, this mimises EMC emission.

Scanned by TapScanner

as frequency is multiplied inside ARM chip. PLL allows changing frequency dynamically. In LPC213X microcontroller, we have one PLL which provides programmable frequencies to CPU.

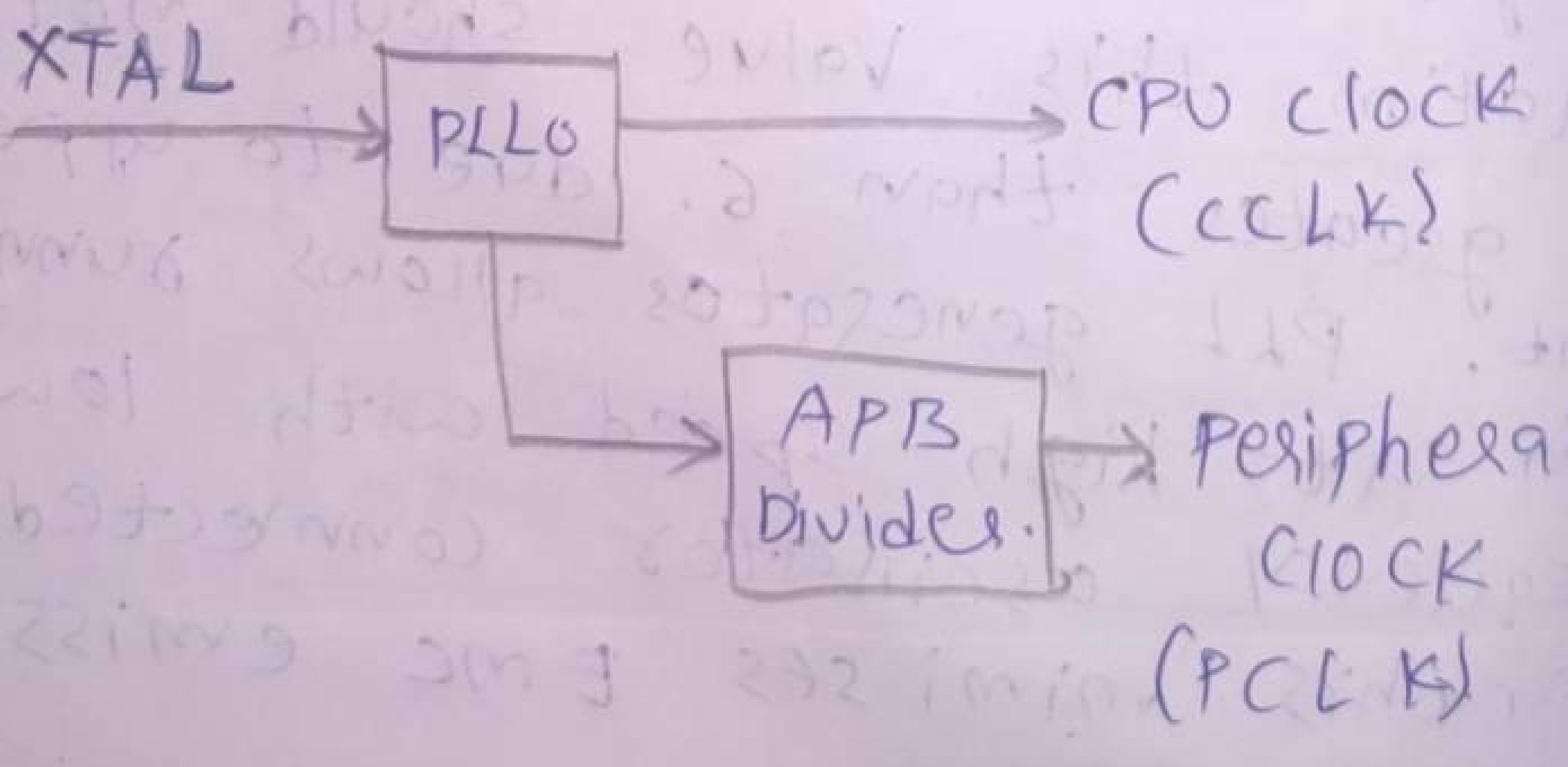
ARM7 LPC213X microcontroller needs 2 clocks:-

* one is for its peripheral.
* other is for its CPU.

→ CPU works faster with higher frequencies whereas peripherals need lower frequencies to work.

→ The peripheral clock & CPU clock gets clock input from PLL (or) external source.

→ After RESET configuration of PLL & VPB (VLSI peripheral Bus) Divider would be first thing to do.



Scanned by TapScanner

=> PLL unit itself uses CCO (current controlled oscillator) which operates in the range between 156- 320 . mHz, so there is additonal divider which keeps CCO within its range, while PLL provides desired frequency. output clock is generated by dividing CCO frequency by 2, 4, 8, 16. minimum divider is '2', so output PLL will always have a duty cycle of 50% for sure.

=> The peripheral clock (PCLK) is derived from CPU clock.

* The APB divider decides the operating frequency of PCLK. The i/p to APB divider is CCLK & o/p is PCLK. By default PCLK runs at 0.25 times of CCLK. To control APB divider, we have a register. VPBDIV. The value in this register controls the generation of PCLK from CCLK.

Scanned by TapScanner

VPBDIV:- 0XD0, APB bus clock is 1/4th of CPU clock

VPBDIV: 0X01, PCLK is same as CCLK

VPBDIV:- 0X02, PCLK is 1/2 of CCLK

VPBDV:- 0X03, Reserved :- {has no effect}.

* PLL Registers :-

NAME	DESCRIPTON.
PLLCON	PLL control register. Holding register for uploading PLL control bits
PLLCFG	PLL configuration register, Holding register for updating PLL configuratn values.
PLLSTAT	PLL status register. Read book register for PLL control & configuratn informatn.
PLLFEED	PLL feed register. This register enables loading of PLL control & configuratn information from PLLCON, PLL CFG into shadow registers that actually affect PLL operation.

Scanned by TapScanner

PLL Programming :-

while configuring clock & PLL. we have to follow these general steps :-

→ select desired operating frequency for your system CCLK.

→ check the oscillator connected to the oscillator on-board (XTAL)

→ calculate the value of PLL divider 'p'. such that FCCO is in the range of 156 - 320MHz.

FCCO = CCLK × 2 × P

→ write the PLLFEED values 0XAA & 0X55.

→ wait till PLL gets Lock

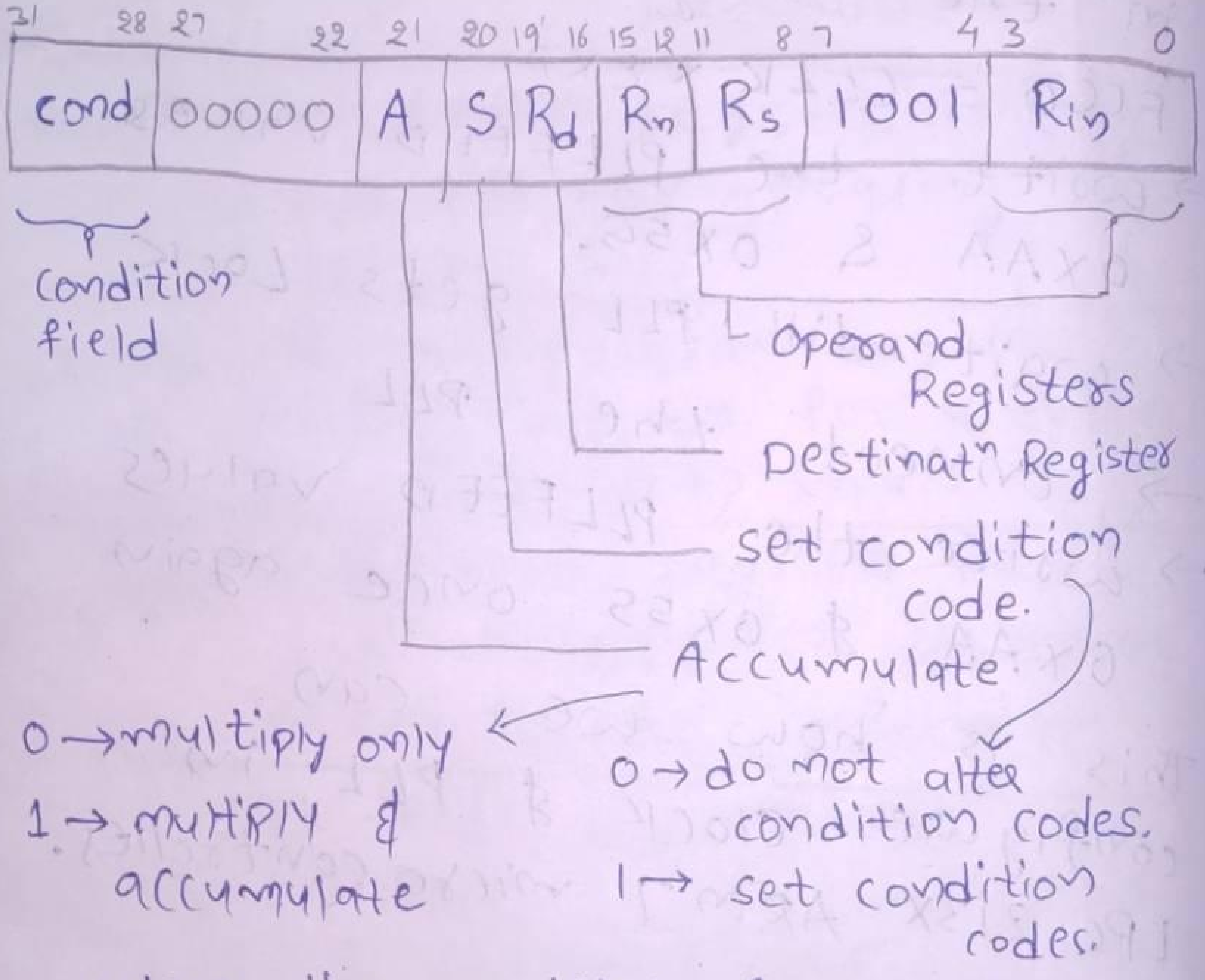
→ connect the PLL

→ write the PLLFEED values 0XAA & 0X55 once again

This is how we can configure clock & PLL in LPC 213X ARM7 micro controller.

5) MLA :-

we will compare multiply (MUL) and multiply Accumulate (MLA) to set a better understanding. These instructions are executed if the condition is true. Both of these instructions uses an 8-bit Booth's algorithm to perform integer multiplication.



While the multiply from the instruction gives $R_d = R_m * R_s$, R_n is figured & should be set to '0' for compatibility with possible future upgrades to the instruction set, the multiply Accumulate gives

$R_d := R_{m} * R_s + R_n$, which can save explicit ADD instruction in some circumstances.

Both forms of the instructions work on operands which may be signed or unsigned integers. The results of signed & unsigned multiply of 32-bit operands differ only in upper 32-bits, whereas lower 32- bits are similar.

The destination register R_d must not be the same as the operand register R_m. R_{15} must not be used as an operand or as the destination register. All other register combinations will give correct results and R_d, R_n & R_s may use the same Register.

While register setting the CPSR flags is optional & is controlled by the S-bit in the instruction.

The N & Z flags are set correctly on the result. The 'C' [carry] flag is set to a meaningless value & the V(overflow) flag is unaffected.

Scanned by TapScanner

MUL takes 1S + (mI) & MLA takes 1S + (m+1) I cycles to execute.

where,

S :- sequential cycle, where ARM7 requests a transform to/from an address which is either the same as the address in the preceeding cycle or is one word after the preceeding cycle.

I :- Internal cycle, where ARM7 does not require a transfer, as it is performing an internal function & no useful prefetching can be performed at the same time.

* m is the number of 8-bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Possible values of Rs

→ 1 : if bits [32:8] of multiplier operand are 0's or 1's.

→ 2 : if bits [32:16] of multiplier operand are 0's / 1's

→ 3 : if bits [32:24] of multiplier operand are 0's / 1's

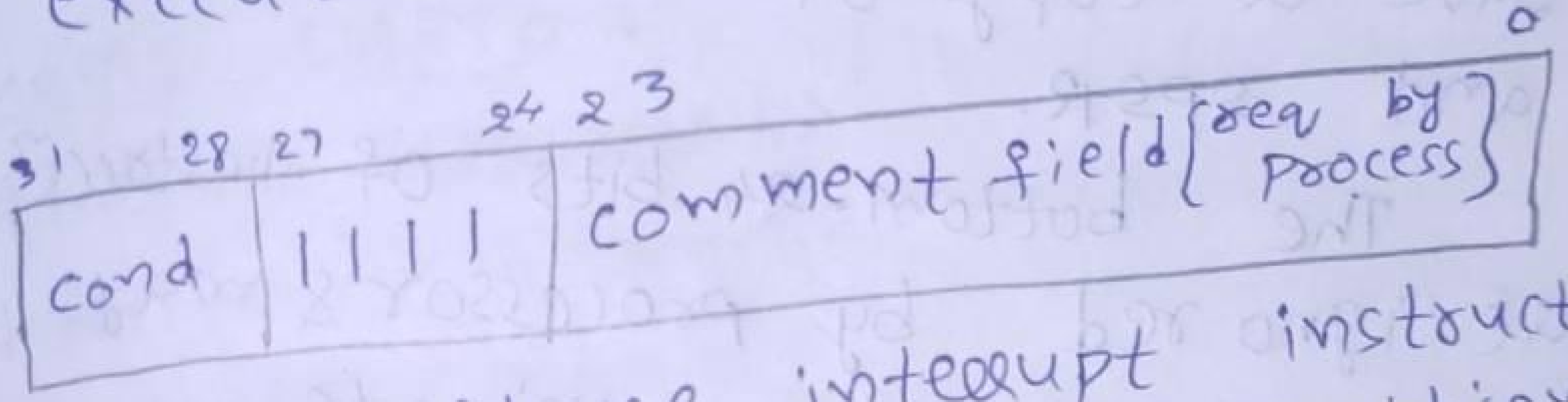
→ 4: in all other cases.

SYNTAX :- MUL {cond} {S} R_d, R_n, R_s

MLA {cond} {S} R_d, R_n, R_s, R_m

2 character condition mnemonic → set condition codes is S present → Expressions evaluating to a Register number other than R15.

* SWI :- software interrupt is only executed if the condition is true.



The software interrupt instruct^n is used to enter super position mode in controlled manner. The instruction causes the software interrupt trap/to be taken, which effects the mode change. The PC is then forced to a fixed value [0x08] & the CPSR saved in SPSR_SVC. If the SWI vector address is suitably protected from modification by the user, a fully protected operating system may be constructed.

The PC is saved in R14_SVC upon entering the software interrupt mode, with the PC adjusted to point to the word after SWI instruction "MOVC PC, R14_SVC", will return the calling program and returns CPSR. Note that the link mechanism is not re-extract, so if the superposition code wishes to use software interrupts with itself, it must save a copy of return address and SPSR.

The bottom-24 bits of instruct? are ignored by processor & may be used to communicate informatn to supervisor code. For instance, supervisor may look at this field & use it to index into an array of entry points for routines which perform various supervisor functions.

software interrupt instruction takes 2S+1N incremental cycles where:

N => Non-sequential cycles, where ARM7 requests a transfer to/from an address which is unrelated to address used in preceeding cycle.

S => ~~Prev~~ Previously stated.

Scanned by TapScanner

SYNTAX:-

$$\begin{tikzcd}[column sep=small]
\text{SWI \{cond\}} & \text{<expression>} \\
\text{2 character condition mnemonic} \arrow[u] & \text{evaluated \& placed in comment field [ignored by ARM7]} \arrow[u]
\end{tikzcd}$$

6) code (for transmitting & reciving chars at 9600 baud):-

using UART0 :-

```
#include <lpc2138.h>
#include <iostream.h>
#include "stdutis.h"
#include "systemInit.h"
 /* macro definition */
#define  SBIT_WordLength  0x00u
#define  SBIT_DLAB   0x07u //Divisor
                 //Address Latch bit.
#define  SBIT_FIFO  0x00u //first in
                 //first out

#define  SBIT_RxFIFO  0x01u
#define  SBIT_Tx FIFO  0x02u
#define  SBIT_RDR   0x00u //Receive data
                          //ready
#define  SBIT_THRE  0x05u// Transmit
          //holding register empty
#define  TX0_PINSEL  0 // Pin select
#define  RX0_PINSEL  2
```

```
char ch;
char tx-data-byte, rx-data-byte;

/* Function to transmit a char */
void uart_TxChar (char ch)
{
  while (util IsBitCleared (U0SLR, SBIT_THRE));
  // wait for previous transmission
  // to be started
  U0THR = ch; // Load the data to
              // be transmitted

}
/* Function to receive a char */
char uart_RxChar()
{
  while (util_IsBitCleared (U0SLR, SBIT_RDR));
  // wait till the data is received
  rx_data_byte = U0RBR;
  // Read received data.
  return rx_data_byte;

}
```

```
/* Function to initialize the UART0 */
   /* at specific baud rate */

void uart_init(unit32 baudrate)

{
 PINSEL0 |= (1<< RX0_PINSEL) | (1<< TX0_
                                  PINSEL);
 // Configure P0.0/P0.1 as RXD0 & TXD0
 U0FCR |= (1<< SBIT_FIFO) | (1<< SBIT_RXFIFO) |
          (1<< SBIT_TXFIFO);
 // Enable FIFO and reset   Rx/TX
 // FIFO buffers.
 U0LCR |= (0X03 << SBIT_WordLength) |
          (1<< SBIT_DLAB);
 // 8-bit data, 1-stop bit, No parity
 // Set DLAB (enable write operation
 // to DLM and DLL)

 U0DLL = 0X00;
 U0DLM = 0X82;
 U0LCR = 0 << SBIT_DLAB;
 // Clear DLAB after setting DLL, DLM for
 // normal operation (disable write
 // operation to DLM and DLL)

}
```

```
int main()
{
 char a[] = "\n 18EC1002 \n";
 // "or" for ASCII strings
 unit16 i; // 16-bits variable
            // (signed integer)
 uart_init.(9600); // initialize the UART0
 // for 9600 baud rate (setting the
 // configuration through register
 // values).
 i = sizeof (a); // Calculate the length
                  // size of the array.
 for(i=0;a[i];i++) // Transmit a
                   // predefined sting
 {
  uart_TxChar (a[i]);
 }
 while (1)
 {
 // Finally receive a char & transmit
 // it infinitely (echo character)
  tx_data_byte = uart_RxChar();
  uart_TxChar (tx_data_byte);
 }
}
```