Advanced Micro-controllers

Agenda

- Q & A
- ARM Introduction
- ISA vs Core
- ARM ISAs
    - Word, Half-word, Byte
    - Instruction set: ARM vs Thumb vs Thumb-2
    - ARM states: ARM, Thumb, Jazelle
    - Unaligned memory access
    - ~~SIMD instruction~~
    - ~~Multi-core architecture - Physical and Logical cores~~
    - ~~Cache (instruction, data, unified | private, shared)~~
    - ~~TrustZone~~

Q & A

1. Hardware vs Software vs Firmware?
    - Hardware: Wires, Electronic Components, Chips, PCBs, etc --> Electronic circuit.
    - Software: Programs that can be easily installed/uninstalled on OS e.g. Chrome, MS Office, ...
    - Firmware: Programs that are fixed/burned in (controller's) ROM/Flash. Firmwares can be modified (but not so easily).
2. What is Programmer?
    - Programmer is hardware and software that is used to burn firmware in Micro-controller's flash.
    - Hardware: STLink-V2, ...
    - Software: STM32CubeProgrammer, FlashMagic, ...
3. Pointer arithmetic

```
// demo05.c
char *p = NULL;
short *q = NULL;
float *r = NULL;
double *s = NULL;
printf("%u, %u, %u, %u\n", ++p, ++q, ++r, ++s);
//output: 1, 2, 4, 8
```

4. Structure member offsets.

```
// demo06.c
#pragma pack(1)

struct test {
    int a;
    short b;
    char c;
};

struct test t1 = { 1234, 12, 'A' };
struct test *p1 = &t1; // assume that &t1 = 400
printf("%u, %u, %u, %u\n", p1, &p1->a, &p1->b, &p1->c);
//output: 400, 400, 404, 406
struct test *p = NULL;
printf("%u, %u, %u, %u\n", p, &p->a, &p->b, &p->c);
//output: 0, 0, 4, 6 <-- offsets of a, b, c
printf("%u, %u, %u, %u\n", ((struct test *)0), &((struct test *)0)->a, &((struct test *)0)->b, &((struct test
*)0)->c);
//output: 0, 0, 4, 6 <-- offsets of a, b, c

#define offset_of(type,member) ((int)&((type*)0)->member)
printf("%u, %u, %u\n",
           offset_of(struct test, a),
```

```
            offset_of(struct test, b),
            offset_of(struct test, c)
        );
//output: 0, 4, 6 <-- offsets of a, b, c
```

5. Slack bytes / Structure padding

```
struct test {
    char a;     // 1 byte
                // 1 slack byte
    short b;    // 2 bytes
    short c;    // 2 bytes
                // 2 slack bytes
    int d;      // 4 bytes
};

struct test t1 = { 1, 2, 3, 4 };

printf("size=%u\n", sizeof(struct test)); // 12 bytes
```

```
#pragma pack(1)
struct test {
    char a;     // 1 byte
    short b;    // 2 bytes
    short c;    // 2 bytes
    int d;      // 4 bytes
};

struct test t1 = { 1, 2, 3, 4 };
printf("size=%u\n", sizeof(struct test)); // 12 bytes
printf("%c\n", t1.a);
```

```
printf("%hd\n", t1.b);
printf("%hd\n", t1.c);
printf("%d\n", t1.d);
```

Word size

- Word size depends on the CPU architecture
- Word size = Amount of data that can be processed at a time by CPU (defined by register sizes).
- On 8086 architecture
  - word = 16 bit (2 bytes)
  - dword = 32 bit (4 bytes)
  - byte = 8 bit (1 byte)
- On ARM architecture
  - word = 32 bit (4 bytes)
  - half-word = 16 bit (2 bytes)
  - byte = 8 bit (1 byte)
  - dword = 64 bit (8 bytes) -- ARM v8

ARM instructions

- v1, v2, v3, v4 -- All instructions are of 32-bit each.
- v4T -- Added Thumb instruction set -- Each instruction is 16 bit.
- ARM instruction set
  - Each instruction is 32-bit.
  - Can access all resources and perform all possible operations -- No limitations.
- Thumb instruction set
  - Each instruction is 16-bit.
  - Limitations
    - Few Thumb instructions cannot access r8-r12 registers.
    - Status register instructions are not allowed (MSR, MRS).
    - Do not support conditional execution.

        - Do not support inline barrel shifter operation.
        - Cannot perform exception handling in Thumb instructions.
    - Increases code density (i.e. number of instructions in unit program memory)
    - Program memory utilization is 70%.
- Thumb-2 instruction set
    - Added in ARM v7 -- will be discussed later.

ARM processor states

- ARM state (T=0 in psr)
    - To execute ARM instructions (4 byte).
    - PC is incremented by 4 bytes (after each instruction fetch)
    - PC is effectively 30 bits -- [31:2] -- Lower 2 bits [1:0] are considered 00.
- Thumb state (T=1 in psr)
    - To execute Thumb instructions (2 byte).
    - PC is incremented by 2 bytes (after each instruction fetch)
    - PC is effectively 31 bits -- [31:1] -- Lower 1 bit [0] is considered 0.
- Jazelle state (J=1 in psr)
    - To execute Java byte code instructions
    - Explained below.

ARM-Thumb interworking

- Few functions can be implemented in ARM instruction set while others are implemented in Thumb instruction set.
- ARM function can call a Thumb function and vice-versa using special instructions.
    - BX func -> Branch with eXchange state
        - Jump to the func
        - Change the T bit = func -- bit0
    - BLX func -> Branch with Link with eXchange state
        - Jump/Call to the func
        - Keep address of next instruction in LR register

    - Change the T bit = func -- bit0

- Example1: arm func1 --> thumb func2

```
.arm
func1:
    ...
    blx func2 | 1   // will make T=1

.thumb
func2:
    ...
```

- Example2: thumb func3 --> arm func4

```
.thumb
func3:
    ...
    blx func4   // will make T=0

.arm
func4:
    ...
```

Jazelle core

- Java codes are compiled into an intermediate code called as byte code.
- Each byte-code instruction (basic) is 8-bit (1 byte).
- These byte code instructions are converted into machine code by JVM and then executed on a processor.
    - Hello.java --> Java compiler --> Hello.class (byte code) --> JVM --> Machine code --> Processor
- On PC, JVM is a software (installed on Windows/Linux/Mac/...).

- Java follows WORA i.e. Write Once Run Anywhere. The same compiled code (.class) can work on different platforms/architectures.
- The JVM functionalities are built into hardware called "Jazelle Core". There are two main types:
  - RCT - Runtime Compilation Target
  - DBX - Direct Bytecode eXecution
- ARMv5TEJ supports Jazelle DBX. A new state added in processor called as Jazelle state.
- Jazelle state (T=0 and J=1).
  - To execute Java byte code instructions.
  - PC is incremented by 4 bytes (after each fetch).
  - Each fetch operation gets multiple byte-code instructions from flash.
  - PC is effectively 30 bits -- [31:2] -- Lower 2 bits [1:0] are considered 00.
- Jazelle core speed up Java byte code execution.

ARM Evolution/History

- ARM v4
  - Half-word and Signed Half-word and Byte support
  - System mode
  - ARM instructions --> Decoder --> Execute
- ARM v4T
  - Added Thumb instructions/state
  - ARM instructions --> Decoder --> Execute
  - Thumb instructions --> Thumb Decoder --> ARM instructions --> Decoder --> Execute
- ARM v5TEJ
  - Added Enhanced DSP support
    - USAT/SSAT -- Saturated Maths
    - MLA -- Multiply and Accumulate
  - Added Jazelle state - to execute Java byte code.