Politecnico di Milano
SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Advanced Operating Systems
A.A. 2019-2020 – Exam date: January, 16th 2020

Prof. William FORNACIARI

Surname (readable)..... Name (readable).....

Matr..... Signature.....

Q1	Q2	TOT

NOTES

It is forbidden to refer to texts or notes of any kind as well as interact with their neighbors. Anyone found in possession of documents relating to the course, although not directly relevant to the subject of the examination will cancel the test. It is not allowed to leave during the first half hour, the task must still be returned, even if it is withdrawn. The presence of the writing (not delivered) implies the renunciation of any previous ratings.

Question 1 (10 points)

Multi-tasking programming may introduce some pitfalls due to the usage of locking mechanisms. Considering the typical real-time systems requirements…

1. What are the pitfalls?
2. How can they occur?
3. Briefly describe the possible solutions shown in class. Make a comparison among solutions addressing the same problem, highlighting PROs/CONs or limitations of each of them.

Question 2 (13 points)

The AES encryption standard is a symmetric block cipher that can encrypt fixed size data blocks of 16 bytes, with different key sizes.
A microcontroller is equipped with an AES peripheral that implements in hardware the AES encryption scheme. Following is reported the documentation of the AES peripheral registers.

Bit access modes:

- **u** = undefined, writing has no effect, reading returns zero
- **w** = always reads as zero, writing 1 causes the corresponding action
- **rw** = software reading and writing is allowed
- **rc-w1** = bit is readable. Writing 0 has no effect, writing 1 clears the bit

AES_CTRL: Address: 0x40000000, Initial value upon reset: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										DONE	EN	DEC	GO	IRQEN	KSZ
u	u	u	u	u	u	u	u	u	u	rc_w1	rw	rw	w	rw	rw

- KSZ: specify key size. 0 means 128 bit key, 1 is for 256 bit key. This bit must be configured BEFORE writing the key to the key registers.
- IRQEN: setting this bit to 1 causes an interrupt function named void AES_irq(); to be called when encryption/decryption of a block is complete.
- GO: writing 1 to this bit starts the encryption/decryption of the block that has been written in the block registers
- DEC: if this bit is 1, starting operation with GO initiates a decryption, if this bit is 0 an encryption will be performed
- EN: this bit must be set first to enable the peripheral
- DONE: this bit is set in hardware after GO has been set and the block encryption/decryption is complete. It must be reset by software before writing a new block to the block registers

AES_BLOCK0: Address: 0x40000010, Initial value upon reset: undefined
This 32 bit register is readable/writable in software and contains the first 4 bytes of the block to be encrypted/decrypted. After DONE has been set, this register contains the first 4 bytes of the encrypted/decrypted block.
AES_BLOCK1: Address: 0x40000014, Initial value upon reset: undefined
Same as AES_BLOCK0, contains bytes 4..7 of the block.
AES_BLOCK2: Address: 0x40000018, Initial value upon reset: undefined
Same as AES_BLOCK0, contains bytes 8..11 of the block.
AES_BLOCK3: Address: 0x4000001c, Initial value upon reset: undefined
Same as AES_BLOCK0, contains bytes 12..15 of the block.
AES_KEY0: Address: 0x40000020, Initial value upon reset: undefined
This 32 bit register is readable/writable in software and contains the first 4 bytes of the AES key to be used for encryption/decryption. The register content is preserved across encryptions/decryptions so if a new encryption/decryption with the same key is required, there is no need to write again the same key.
AES_KEY1: Address: 0x40000024, Initial value upon reset: undefined

Same as AES_KEY0, contains bytes 4..7 of the key.
AES_KEY2: Address: 0x40000028, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 8..11 of the key.
AES_KEY3: Address: 0x4000002c, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 12..15 of the key.
AES_KEY4: Address: 0x40000030, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 16..19 of the key. Only for 256 bit keys.
AES_KEY5: Address: 0x40000034, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 20..23 of the key. Only for 256 bit keys.
AES_KEY6: Address: 0x40000038, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 24..27 of the key. Only for 256 bit keys.
AES_KEY7: Address: 0x4000003c, Initial value upon reset: undefined
Same as AES_KEY0, contains bytes 28..31 of the key. Only for 256 bit keys.

You are required to:

1. Write **data structures** and **macros** which could be used to access these peripheral registers, by a C/C++ coded program, using a syntax similar to AES->CTRL;
2. Implement the following functions:

```
/**
 * Initializes the peripheral and set the key that will be used for future
encryptions/decryptions
 * \param key pointer to the key
 * \param key_size 128 for a 128 bit key, 256 for a 256 bit key
 */
void AES_init_key(const unsigned int key[], int key_size);

/**
 * Blocking function that returns once all bytes are encrypted.
 * \param dest encrypted data have to be stored here
 * \param src pointer to data to encrypt
 * \param len number of 16 byte blocks to encrypt
 */
void AES_encrypt(unsigned int dest[], const unsigned int src[], int len);

/**
 * Blocking function that returns once all bytes are decrypted.
 * \param dest decrypted bytes have to be stored here
 * \param src pointer to bytes to decrypt
 * \param len number of 16 byte blocks to decrypt
 */
void AES_decrypt(unsigned int dest[], const unsigned int src[], int len);
```

Note: it is expected that the encrypt and decrypt functions will be implemented using polling, as no OS context switch or CPU sleep facility is provided.

Solution:

```
struct AES_STRUCT
{
    volatile unsigned short CTRL;
    unsigned char padding1[14];
    volatile unsigned int BLOCK0;
    volatile unsigned int BLOCK1;
    volatile unsigned int BLOCK2;
    volatile unsigned int BLOCK3;
    volatile unsigned int KEY0;
    volatile unsigned int KEY1;
    volatile unsigned int KEY2;
    volatile unsigned int KEY3;
    volatile unsigned int KEY4;
    volatile unsigned int KEY5;
    volatile unsigned int KEY6;
    volatile unsigned int KEY7;
};

#define AES ((struct AES_STRUCT*)0x40000000)

void AES_init_key(const unsigned int key[], int key_size)
{
    AES->CTRL |= 1<<4;

    if(key_size == 256) AES->CTRL |= 1<<0;
    else AES->CTRL &= ~(1<<0);

    AES->KEY0 = key[0];
    AES->KEY1 = key[1];
    AES->KEY2 = key[2];
    AES->KEY3 = key[3];

    if(key_size == 256)
    {
        AES->KEY4 = key[4];
        AES->KEY5 = key[5];
        AES->KEY6 = key[6];
        AES->KEY7 = key[7];
    }
}

// Helper function to avoid code duplication
static void encdec_helper(unsigned int dest[], const unsigned int src[], int len)
{
    for(int i = 0; i < len; i++)
    {
        AES->BLOCK0 = src[4*i + 0];
        AES->BLOCK1 = src[4*i + 1];
        AES->BLOCK2 = src[4*i + 2];
        AES->BLOCK3 = src[4*i + 3];

        AES->CTRL |= (1<<2);
        while((AES->CTRL & (1<<5))==0) ; /* poll */
        AES->CTRL |= (1<<5);

        dest[4*i + 0] = AES->BLOCK0;
        dest[4*i + 1] = AES->BLOCK1;
        dest[4*i + 2] = AES->BLOCK2;
        dest[4*i + 3] = AES->BLOCK3;
    }
}

void AES_encrypt(unsigned int dest[], const unsigned int src[], int len)
{
    AES->CTRL &= ~(1<<3);

    encdec_helper(dest,src,len);
}
```

```
void AES_decrypt(unsigned int dest[], const unsigned int src[], int len)
{
    AES->CTRL |= 1<<3;

    encdec_helper(dest,src,len);
}
```