4.1 Secure Hash Algorithm Accelerator

1. Experiment purpose

In this lesson, we mainly learn function of Secure Hash Algorithm Accelerator of K210.

2.Experiment preparation

2.1 components

Core 0 and Core 1 in K210 chip

2.2 Component characteristics

The SHA256 accelerator is a computing unit used to calculate SHA-256.

2.3 SDK API function

The header file is sha256.h

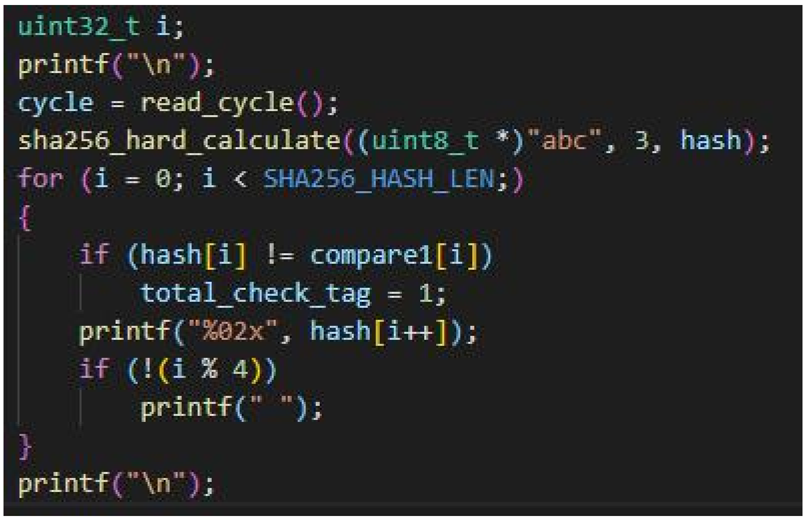
Support SHA-256 calculation.

We will provide users with the following interfaces:

• sha256_init: Initialize the SHA256 accelerator peripheral.
• sha256_update: Pass in a data block to participate in SHA256 Hash calculation.
• sha256_final: End the SHA256 Hash calculation of the data.
• sha256_hard_calculate: Calculate its SHA256 Hash for continuous data at one time.

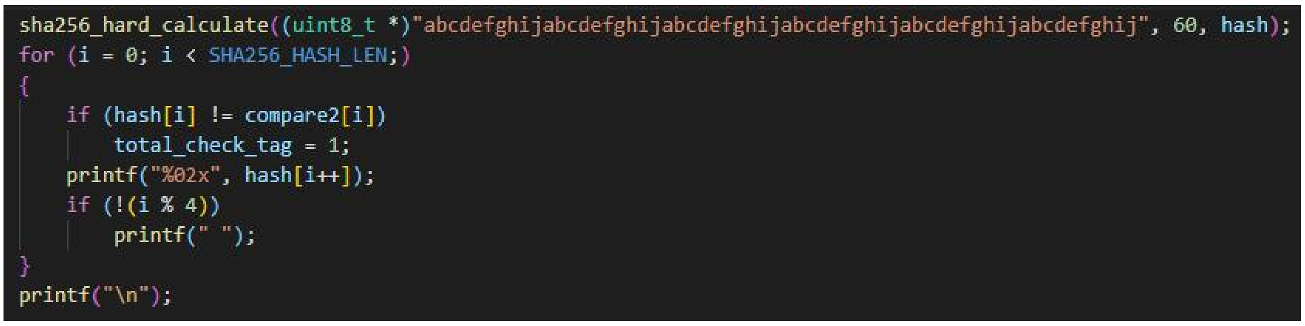
3. Experiment procedure

3.1 Calculate the hash value of the string 'abc' and print it out.



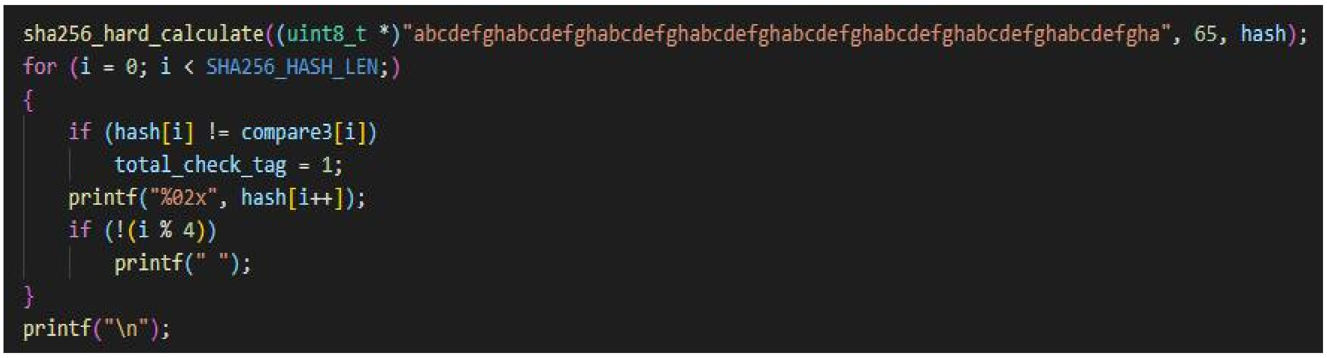
```
uint32_t i;
printf("\n");
cycle = read_cycle();
sha256_hard_calculate((uint8_t *)"abc", 3, hash);
for (i = 0; i < SHA256_HASH_LEN;)
{
    if (hash[i] != compare1[i])
        total_check_tag = 1;
    printf("%02x", hash[i++]);
    if (!(i % 4))
        printf(" ");
}
printf("\n");
```

3.2 Calculate the hash value of the string 'abcdefghabcdefghabcdefghabcdefghabcdefghabcdefghabcdefghabcdefgha' and print it out.



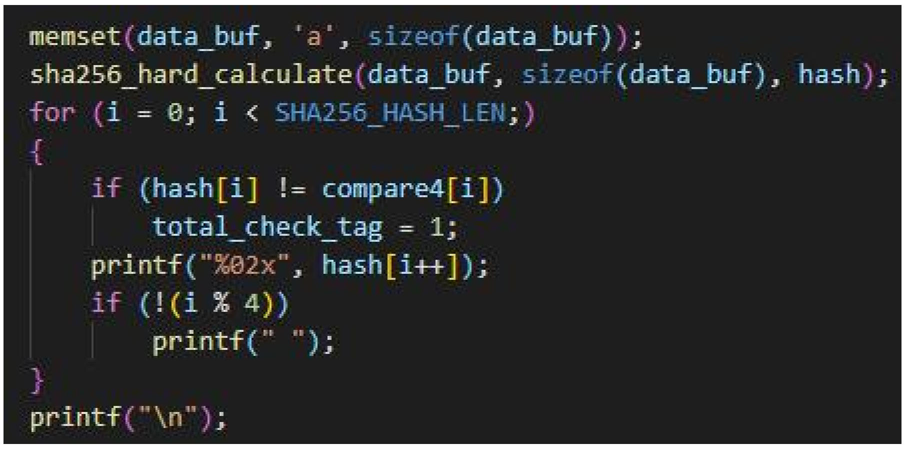
```c
sha256_hard_calculate((uint8_t *)"abcdefghijabcdefghijabcdefghijabcdefghijabcdefghijabcdefghij", 60, hash);
for (i = 0; i < SHA256_HASH_LEN;)
{
    if (hash[i] != compare2[i])
        total_check_tag = 1;
    printf("%02x", hash[i++]);
    if (!(i % 4))
        printf(" ");
}
printf("\n");
```

3.3 Calculate the hash value of the string ‘abcdefghijabcdefghijabcdefghijabcdefghijabcdefghijabcdefghij’and print it out.



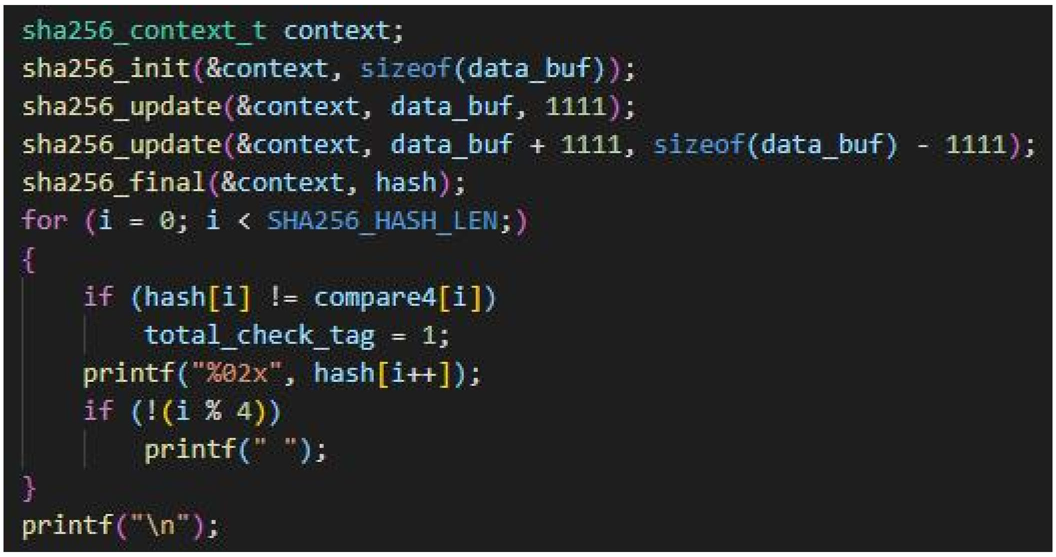
```c
sha256_hard_calculate((uint8_t *)"abcdefghabcdefghabcdefghabcdefghabcdefghabcdefghabcdefghabcdefgha", 65, hash);
for (i = 0; i < SHA256_HASH_LEN;)
{
    if (hash[i] != compare3[i])
        total_check_tag = 1;
    printf("%02x", hash[i++]);
    if (!(i % 4))
        printf(" ");
}
printf("\n");
```

3.4 Calculate the hash value of multiple characters ‘a’ and print it out.

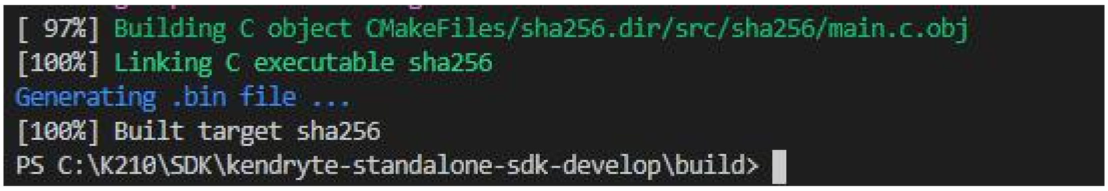


```c
memset(data_buf, 'a', sizeof(data_buf));
sha256_hard_calculate(data_buf, sizeof(data_buf), hash);
for (i = 0; i < SHA256_HASH_LEN;)
{
    if (hash[i] != compare4[i])
        total_check_tag = 1;
    printf("%02x", hash[i++]);
    if (!(i % 4))
        printf(" ");
}
printf("\n");
```

3.5 Calculate the hash value of multiple characters ‘a’ in blocks and print it out.

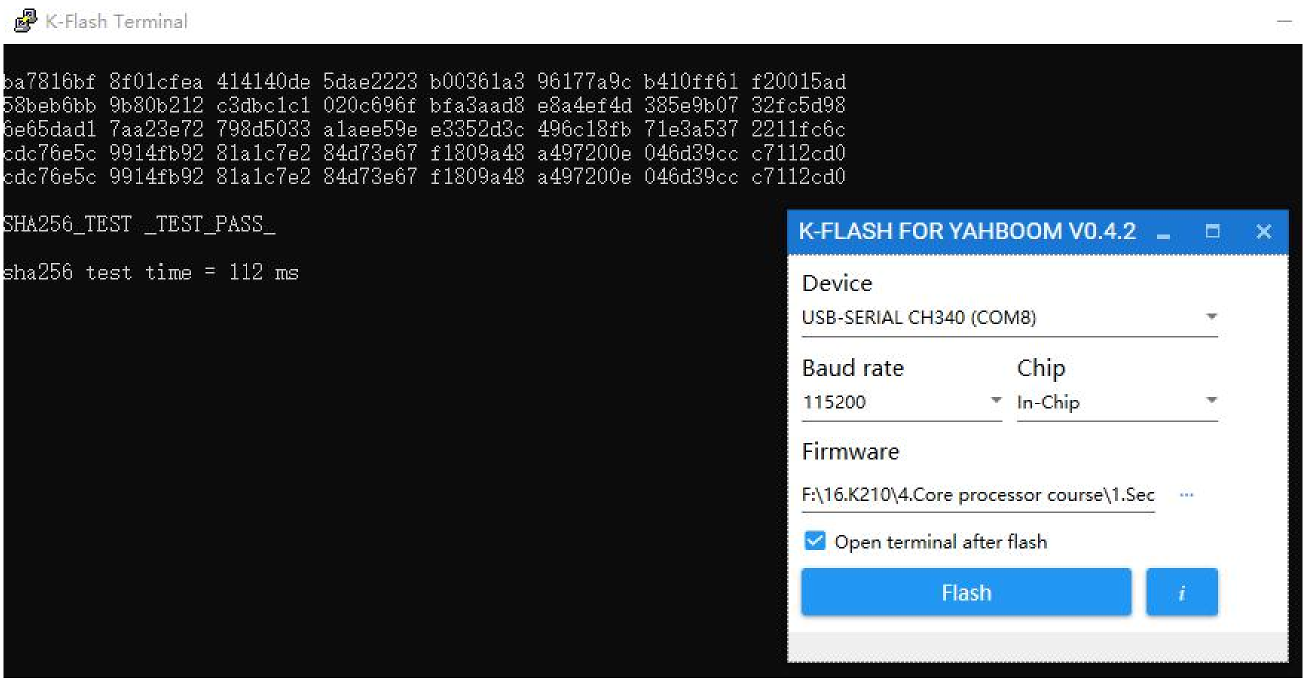

```
sha256_context_t context;
sha256_init(&context, sizeof(data_buf));
sha256_update(&context, data_buf, 1111);
sha256_update(&context, data_buf + 1111, sizeof(data_buf) - 1111);
sha256_final(&context, hash);
for (i = 0; i < SHA256_HASH_LEN;)
{
    if (hash[i] != compare4[i])
        total_check_tag = 1;
    printf("%02x", hash[i++]);
    if (!(i % 4))
        printf(" ");
}
printf("\n");
```

3.6 Compile and debug, burn and run
Copy the sha256 to the src directory in the SDK.
Then, enter the build directory and run the following command to compile.
cmake .. -DPROJ=sha256 -G "MinGW Makefiles"
make


```
[ 97%] Building C object CMakeFiles/sha256.dir/src/sha256/main.c.obj
[100%] Linking C executable sha256
Generating .bin file ...
[100%] Built target sha256
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **sha256.bin** file will be generated in the build folder.
We need to use the type-C data cable to connect the computer and the K210 development board.
Open kflash, select the corresponding device, and then burn the **sha256.bin** file to the K210 development board.

4. Experimental phenomenon
After the firmware is write, a terminal interface will pop up.It will print out hash value.


K-Flash Terminal

```
ba7816bf 8f01cfea 414140de 5dae2223 b00361a3 96177a9c b410ff61 f20015ad
58beb6bb 9b80b212 c3dbc1c1 020c696f bfa3aad8 e8a4ef4d 385e9b07 32fc5d98
6e65dad1 7aa23e72 798d5033 a1aee59e e3352d3c 496c18fb 71e3a537 2211fc6c
cdc76e5c 9914fb92 81a1c7e2 84d73e67 f1809a48 a497200e 046d39cc c7112cd0
cdc76e5c 9914fb92 81a1c7e2 84d73e67 f1809a48 a497200e 046d39cc c7112cd0

SHA256_TEST _TEST_PASS_

sha256 test time = 112 ms
```

K-FLASH FOR YAHBOOM V0.4.2

Device: USB-SERIAL CH340 (COM8)

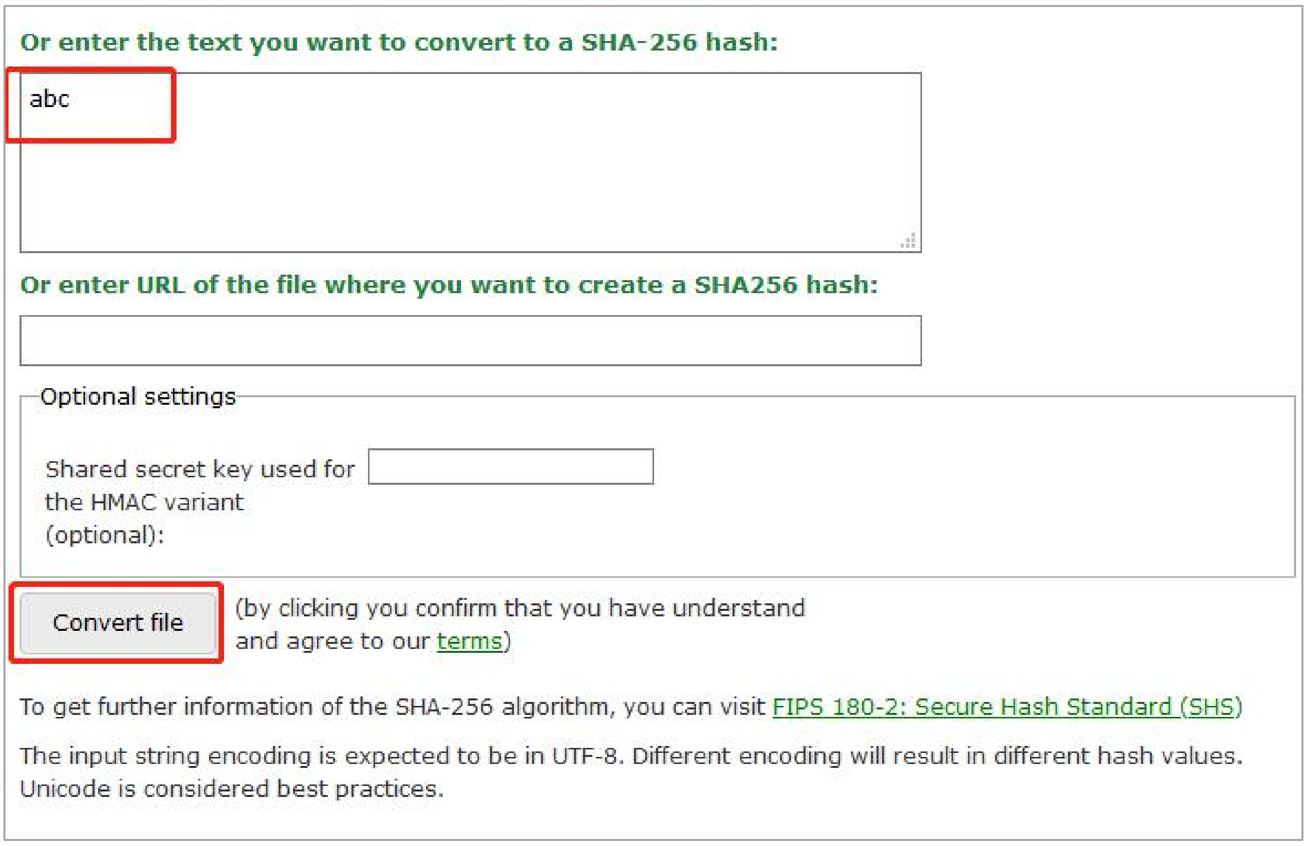
Baud rate: 115200

Chip: In-Chip

Firmware: F:\16.K210\4.Core processor course\1.Sec

☑ Open terminal after flash

Enter the following website to verify the correctness of the hash value.
https://hash.online-convert.com/sha256-generator
Enter the string you want to convert in the box below, and then click **"Convert file"**.
Wait patiently for completion.


Or enter the text you want to convert to a SHA-256 hash:

abc

Or enter URL of the file where you want to create a SHA256 hash:

Optional settings

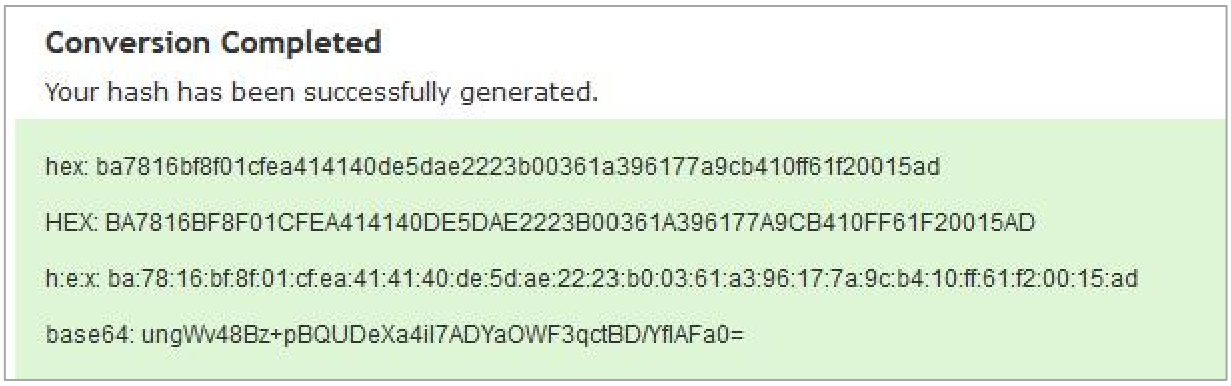
Shared secret key used for the HMAC variant (optional):

(by clicking you confirm that you have understand and agree to our terms)

To get further information of the SHA-256 algorithm, you can visit FIPS 180-2: Secure Hash Standard (SHS)

The input string encoding is expected to be in UTF-8. Different encoding will result in different hash values. Unicode is considered best practices.

After the conversion is successful, you can see the hash value of the digest. Consistent with the results printed by the program.



```
Conversion Completed
Your hash has been successfully generated.

hex: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

HEX: BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD

h:e:x: ba:78:16:bf:8f:01:cf:ea:41:41:40:de:5d:ae:22:23:b0:03:61:a3:96:17:7a:9c:b4:10:ff:61:f2:00:15:ad

base64: ungWv48Bz+pBQUDeXa4iI7ADYaOWF3qctBD/YflAFa0=
```

5. Experiment summary

5.1 SHA256 is a member of the SHA-2 encryption system, and the basic encryption algorithms of all SHA-2 members are the same, but the length of the generated digest and the number of cycles are different.

5.2 The digest generated by SHA256 each time is 256bit.

5.3 SHA256 is a widely used security hash algorithm.

Appendix -- API

Header file is sha256.h

sha256_init

Description: Initialize the SHA256 accelerator peripheral.

Function prototype: **void sha256_init(sha256_context_t *context, size_t input_len)**

Parameter:

Parameter name	Description	Input/Output
context	SHA256 context object	Input
input_len	The length of the message to be calculated for the SHA256 hash	Input

Return value: no

Eg:

sha256_context_t context;
sha256_init(&context, 128U);

sha256_update

Description: Pass in a data block to participate in SHA256 Hash calculation

Function prototype: **void sha256_update(sha256_context_t *context, const void *input, size_t input_len)**

Parameter:

Parameter name	Description	Input/Output
context	SHA256 context object	Input
input	SHA256 calculated data block to be added to the calculation	Input

Parameter name	Description	Input/Output
buf_len	The length of the SHA256 calculation data block to be added to the calculation	Input

Return value: no

sha256_final

Description: End SHA256 Hash calculation of data

Function prototype: **void sha256_final(sha256_context_t *context, uint8_t *output)**

Parameter:

Parameter name	Description	Input/Output
context	SHA256 context object	Input
output	To store the result of SHA256 calculation, it is necessary to ensure that the size of the incoming buffer is 32Bytes or more	Output

Return value: no

sha256_hard_calculate

Description: Calculate its SHA256 Hash for continuous data at one time

Function prototype: **void sha256_hard_calculate(const uint8_t *input, size_t input_len, uint8_t *output)**

Parameter:

Parameter name	Description	Input/Output
input	Data waiting for SHA256 calculation	Input
input_len	Length of data waiting for SHA256 calculation	Input
output	To store the result of SHA256 calculation, it is necessary to ensure that the size of the incoming buffer is 32Bytes or more	Output

Return value: no

Eg:

uint8_t hash[32];

sha256_hard_calculate((uint8_t *)"abc", 3, hash);

Eg:

One-time calculation

sha256_context_t context;

sha256_init(&context, input_len);

sha256_update(&context, input, input_len);

sha256_final(&context, output);

Or you can directly call the function,

sha256_hard_calculate(input, input_len, output);

Perform block calculations

sha256_context_t context;

sha256_init(&context, input_piece1_len + input_piece2_len);

```
sha256_update(&context, input_piece1, input_piece1_len);
sha256_update(&context, input_piece2, input_piece2_len);
sha256_final(&context, output);
```