CS577 Project Report (Phase 1)

kyber768

Vishal Kumar (226101005)
Akash Lal Dutta (226101001)
Sreeparna Das (226101004)
Vivekanada G (226201004)
Soumya Asati (224101046)

Indian Institute of Technology Guwahati
Department of Computer Science and Engineering

Here is our top function `crypto_kem_dec` we need to synthesize with explanations:

```
#include <stddef.h>
#include <stdint.h>
#include "kem.h"
#include "params.h"
#include "rng.h"
#include "symmetric.h"
#include "verify.h"
#include "indcpa.h"
```

This code includes several header files that define some functions and macros used in the code.

```
int crypto_kem_dec(unsigned char *ss,
                   const unsigned char *ct,
                   const unsigned char *sk)
```

This is the implementation of the "crypto_kem_dec" function, which is part of a post-quantum key encapsulation mechanism (KEM). This function takes three arguments: "ss", which is a pointer to a buffer that will contain the shared secret after decryption; "ct", which is a pointer to the ciphertext that should be decrypted; and "sk", which is a pointer to the secret key used for decryption.

```
size_t i;
int fail;
uint8_t buf[2*KYBER_SYMBYTES];        // Buffer for key and coins
uint8_t kr[2*KYBER_SYMBYTES];         // Will contain key, coins
uint8_t cmp[KYBER_CIPHERTEXTBYTES];   // Will contain ciphertext
const uint8_t *pk = sk + KYBER_INDCPA_SECRETKEYBYTES;
```

These lines declare some variables that will be used later in the code. "i" is a variable used for looping "fail".

- `size_t i;`: Declares a variable "i" of type `size_t`.
- `int fail;`: Declares a variable fail of type `int`.
- `uint8_t "buf[2*KYBER_SYMBYTES]";`: Declares an array `buf` of size "2*KYBER_SYMBYTES" that can store unsigned 8-bit integers `(uint8_t)`.
- `uint8_t "kr[2*KYBER_SYMBYTES]";`: Declares an array `kr` of size "2*KYBER_SYMBYTES" that can store unsigned 8-bit integers `(uint8_t)`.
- `uint8_t "cmp[KYBER_CIPHERTEXTBYTES]";`:
  Declares an array `cmp` of size "KYBER_CIPHERTEXTBYTES" that can store unsigned 8-bit integers `(uint8_t)`.
- `const "uint8_t *pk = sk + KYBER_INDCPA_SECRETKEYBYTES";`:
  Declares a constant pointer `pk` to an unsigned 8-bit integer `(uint8_t)` and initializes it to the memory address of "sk plus KYBER_INDCPA_SECRETKEYBYTES". This means that `pk` points to the memory location immediately after the "KYBER_INDCPA_SECRETKEYBYTES" bytes of `sk`.

```
 indcpa_dec(buf, ct, sk);
```

This line uses the "indcpa_dec" function to decode the ciphertext "ct" using the secret key "sk" and store the result in the buffer "buf". This step obtains the message polynomial.

```
for(i=0;i<KYBER_SYMBYTES;i++)
    buf[KYBER_SYMBYTES+i] = sk[KYBER_SECRETKEYBYTES-2*KYBER_SYMBYTES+i];
```

This loop copies the last "KYBER_SYMBYTES" bytes of the secret key "sk"into the second half of the buffer "buf". This step generates a random string "buf" for the key and coins.

```
hash_g(kr, buf, 2*KYBER_SYMBYTES);
```

This line uses the "hash_g" function to hash the entire buffer "buf" and store the result in "kr". This step generates the key and coins.

```
indcpa_enc(cmp, buf, pk, kr+KYBER_SYMBYTES);
```

This line uses the "indcpa_enc" function to encrypt the buffer "buf" using the public key "pk" and the random string "kr+KYBER_SYMBYTES" (which contains the coins). The result is stored in "cmp".

```
fail = verify(ct, cmp, KYBER_CIPHERTEXTBYTES);
```

This line uses the "verify"function to compare the ciphertext "ct" with the encrypted buffer "cmp". If they are equal, "fail" is set to 0 (success). If they are not equal, "fail" is set to 1 (failure).

```
hash_h(kr+KYBER_SYMBYTES, ct, KYBER_CIPHERTEXTBYTES);
```

This line uses the "hash_h" function to hash the ciphertext "ct" and store the result in the second half of "kr" (which contains the coins). This step overwrites the coins with the hash of the ciphertext to protect against potential attacks.

```
cmov(kr, sk+KYBER_SECRETKEYBYTES-KYBER_SYMBYTES, KYBER_SYMBYTES, fail);
```

This line uses the "cmov" to conditionally move the last "KYBER_SYMBYTES" bytes of the secret key "sk" into the first half of "kr". This step is done only if "fail" is not equal to 0, indicating a decryption failure. Otherwise, "kr" remains unchanged.

```
kdf(ss, kr, 2*KYBER_SYMBYTES);
return 0;
```

This line uses the "kdf" function to derive the shared secret "ss" from the entire "kr" buffer (which contains the key and coins) and store it in "ss". Finally, the function returns 0 to indicate success.

Here is the modified code with the indicated modifications:

```
int crypto_kem_dec(
      unsigned char ss[KYBER_CIPHERTEXTBYTES],
      const unsigned char ct[KYBER_CIPHERTEXTBYTES],
      const unsigned char sk[KYBER_CIPHERTEXTBYTES])
```

The function takes arrays "ct", "sk", and "ss" as arguments rather than pointers to these arrays as given in our original code.

```
    for (i = 0; i < KYBER_CIPHERTEXTBYTES; i++) {
        ct_local[i] = ct[i];
    }

    for (i = 0; i < KYBER_SECRETKEYBYTES; i++) {
        sk_local[i] = sk[i];
    }
```

The arrays ct_local and sk_local are created and used instead of the original pointers to "ct" and "sk", and the function loops through the original arrays and copies their values to these local arrays.

```
indcpa_dec(buf, ct_local, sk_local);
```

The function uses the local arrays ct_local and sk_local in the call to "indcpa_dec" instead of the original pointers to "ct" and "sk".

```
for (i = 0; i < KYBER_SYMBYTES; i++) {
        buf[KYBER_SYMBYTES+i] = sk_local[KYBER_SECRETKEYBYTES-2*
           KYBER_SYMBYTES+i];
    }

    hash_g(kr, buf, 2*KYBER_SYMBYTES);

    indcpa_enc(cmp, buf, pk, kr+KYBER_SYMBYTES);

    fail = verify(ct_local, cmp, KYBER_CIPHERTEXTBYTES);

    hash_h(kr+KYBER_SYMBYTES, ct_local, KYBER_CIPHERTEXTBYTES);

    cmov(kr, sk_local+KYBER_SECRETKEYBYTES-KYBER_SYMBYTES, KYBER_SYMBYTES,
        fail);

    kdf(ss, kr, 2*KYBER_SYMBYTES);
}
```

The function uses ct_local instead of "ct" in the call to "verify".

Results obtained from the synthesis of our top function in `Vitis_HLS`

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	24	-
FIFO	-	-	-	-	-
Instance	99	200	20329	143030	0
Memory	17	-	48	25	0
Multiplexer	-	-	-	1033	-
Register	-	-	41	-	-
Total	116	200	20418	144112	0
Available	730	740	269200	134600	0
Utilization (%)	15	27	7	107	0

Figure 1: Utilization Estimates of the original code given to synthesis

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	18	-
FIFO	-	-	-	-	-
Instance	97	200	18728	126280	0
Memory	16	-	112	37	0
Multiplexer	-	-	-	1220	-
Register	-	-	53	-	-
Total	113	200	18893	127555	0
Available	730	740	269200	134600	0
Utilization (%)	15	27	7	94	0

Figure 2: Utilization Estimates of the modified code after synthesis