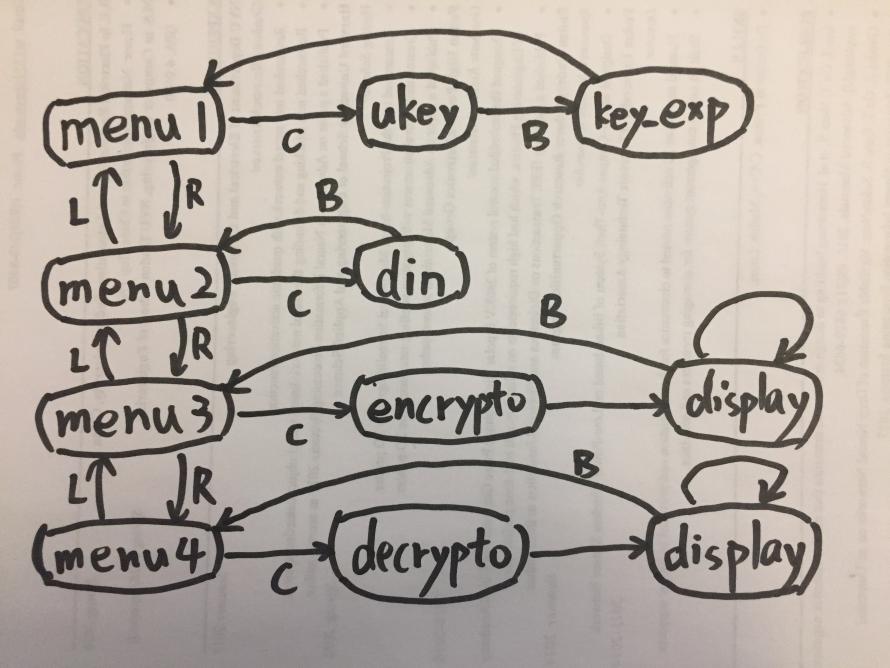
3. RC5 Implementation

3.1 Assmebly code

3.1.1 FSM diagram



This picture shows the diagram of our RC5 code. With the help of the input/output interface (modified ‘reg\_file’ modules), our program supports changing ‘ukey’ and ‘din’ on-the-fly. There are 4 buttons: LEFT(L), RIGHT(R), CONFIRM(C) and BACK(B). User can use these buttons to enter or exit different subprograms.

**UKEY**:

This is the subprogram for sending 128-bit ukey. The ukey will be stored into data memory [50 to 53] in little-endian.

**KEY\_EXP:**

This is the key expansion function. Round-key will be generated based on the ukey in the data memory. The expanded round-key will be stored into data memory [0 to 25].

**DIN:**

This is the subprogram for sending 64-bit input-text. The input-text will be stored into data memory [54 to 55] in little-endian.

**ENCRYPTION:**

This is the encryption function. 64-bit Cypher-text encrypted from input-text will be stored into data memory [30 to 31] in little-endian.

**DECRYPTION:**

This is the decryption function. 64-bit Plain-text decrypted from input-text will be stored into data memory [32 to 33] in little-endian.

**DISPLAY:**

This subprogram will show the output-text on the 7-segment display. User can use LEFT or RIGHT switch from higher or lower 32 bits of

Please check the source ‘rc5\_optimized.asm’ for more details. Our source code is well commented for reviewing.

3.1.2 Assembly to Binary

We wrote a simple compiler in python (load\_instructions.py) to convert our assembly code (\*.asm) to machine code (\*.binary) automatically.

3.1.3 Time complexity analysis

The critical part in the RC5 implementation is how good we can run the key expansion, encryption and decryption in terms of running time. So in this section we will briefly analyze the time complexity of rc5.

The running time of **key expansion** is **O(TL + TS + (Trot + Cexp)\*78 + C0)**, where TL is for L-Initialization, TS is for S-Initialization, Trot + Cexp is for one round of skey-Expansion, C0 is for other constant time operations. The running time of **encryption/decryption** is **O((Trot + Txor + C1)\*2\*12 + C0)**, where (Trot + Txor + C1)\*2 is for one round of encryption/decryption, C0 is for other constant time operations.

Considering that factor 78 is so large, optimizing Trot and Cexp is more important comparing to TL and TS. Following this idea, we will discuss about S-Init and ROTATION optimization later.

**Encryption/Decryption:**

3.1.5 ROTATION optimization

3.1.4 XOR optimization

3.2 Testbench

10 ukey \* 1000 text for functional. 2 ukey \* 3 text for timing.

3.3 Fuctional simulation

screenshots and explanation.

report number of cycles for key\_exp, enc, dec here.

3.4 Timing simulation

screenshots and explanation.

3.5 FPGA Implementation