Programmet läser av knapparna och använder sig av en avkodningsrutin som hittats på nätet. Programmet initierar Timer2 och UART. Användning av UART-kommunikation används i funktionen `writeHexValueToSegmentAtAdress(received, 0xC0);`

Först anropas `clearSegmentsAndLEDs();`
Denna ser ut så här:

```
void clearSegmentsAndLEDs(void){
            sendInstruction(0x8F); //Lights up
            sendInstruction(0x40); //Command, write to chip,autoincrements
            sendStartAdress(0xC0);//Send start adress
            // Sendinging (data,count)
            sendMultipleData(0x00, 16);
}
```

Därefter skrivs sifrrorna 12345678 ut till displayen
`writeIntValueToSegments(0x12345678);`
som ser ut så här:

```
void writeIntValueToSegments(uint32_t value)
{
            uint8_t adress = 0xC0;
            uint8_t extractedNibblesInByteFormat[8];
            uint32_t mask = 0x0F;
            uint32_t extractedNibbleAs32bit;
            uint8_t nibble;
            uint8_t i;
            for(i = 0; i < 8; i++){
                        extractedNibbleAs32bit = mask & value;
                        extractedNibbleAs32bit = extractedNibbleAs32bit >> 4*i;
                        nibble = (uint8_t)extractedNibbleAs32bit;
                        nibble = decodeHexToSevenSegment(nibble);
                        extractedNibblesInByteFormat[i] = nibble;
                        mask = mask << 4;
            }
            for(i = 0; i < 8; i++){
                        sendInstruction(DATA|FIXED);//Command write
autoincrements
                        sendStartAdress(adress +2*i);
                        sendMultipleData(extractedNibblesInByteFormat[7-i], 1); //
                        }
}
```

Därefter nås main-loopen, funktionen `readKeysAndDecodeInternalScan,` borde refaktorera detta dumma namn, gör avkodningen enligt algoritm som hittats på nätet.Detta är inte samma algoritm som tillverkaren använder, i sin exempelkod som skickades till mig.

```
            while (1)
            {

                        //uReceivedData = read4BytesAndConvertToInt();
                        received = readKeysAndDecodeInternalScan();
                        if(received > 0){
                           clearSegmentsAndLEDs();
                           //writeIntValueToSegments(uReceivedData);
                           writeHexValueToSegmentAtAdress(received, 0xC0);

                        }
```

```
                udelay(5000);
        }
}
```

Så här ser den ut:

```
uint8_t readKeysAndDecodeInternalScan()
{
        int i,j;
                uint8_t received = 0;
                uint8_t keys = 0;

                sendInstructionRead();//Read
                DIO(HIGH);
                CDELAY;
        for(i = 0; i < 4; i++ )
        {
                for(j = 0; j < 8; j++)
                {
                        CLK(LOW);
                        CDELAY;
                        CLK(HIGH);

                        if(rDIO){
                                received = received | (0x01 << j);
                        }
                        CDELAY;
                        CDELAY;
                }
                keys |= received << i;
                received = 0;
        }
                CLK(LOW);
                DIO(LOW);
                STB(HIGH);
                CLK(HIGH);
                DIO(HIGH);
                return keys;
}
```

Använder mig av standardmetoden att läsa in en seriell ström, som bit-bangas. Variablen received är initialiserad till noll, och om rDIO läser hög så sätts korresponderande bit genom att loop-indexeringsvariabeln j vänster-skiftar en bit-mask 0x01.
Magin sker på slutet av den yttre for-loopen, där received vänsterskiftas lika många gånger som loop-index-variabeln i och bitvis OR, görs med tidigare värden.

Funktionen returnerar och anrop gör till `writeHexValueToSegmentAtAdress(received, 0xC0);`

```
void writeHexValueToSegmentAtAdress(uint8_t value, uint8_t adress)
{
                uint8_t data;
        //Decoding low nibble to segment display
                 uint8_t lowMask = 0x0F;
                 uint8_t lowNibble;
                 lowNibble= value & lowMask;
                 lowNibble = decodeHexToSevenSegment(lowNibble);
```

```
                    //Decoding high nibble to segment display
                     uint8_t highMask = 0xF0;
                     uint8_t highNibble;
                     highNibble= value & highMask;
                     highNibble = highNibble >> 4 ;
                     highNibble = decodeHexToSevenSegment(highNibble);


                     sendInstruction(DATA|FIXED);//Command, write  no auto
                     sendStartAdress(adress+2);
                     sendMultipleData(lowNibble, 1); // One byte


                     sendInstruction(DATA|FIXED); //No auto increment
                     sendStartAdress(adress);
                     sendMultipleData(highNibble, 1);

                     data = value;
                     switch(data)
                     {
                    case 16:
                    HAL_UART_Transmit(&huart2,"16_",sizeof("16_"),HAL_MAX_DE
                    LAY);
                                break;
                     case 32:

                    HAL_UART_Transmit(&huart2,"32_",sizeof("32_"),HAL_MAX_DE
                    LAY);
                     break;

                    case 64:
                    HAL_UART_Transmit(&huart2,"64_",sizeof("64_"),HAL_MAX_DE
                    LAY);
                    break;
                     case 128:

                    HAL_UART_Transmit(&huart2,"128_",sizeof("128_"),HAL_MAX_
                    DELAY);
                    break;
                     default:
                        data += 0x30;

                    HAL_UART_Transmit(&huart2,&data,sizeof(data),HAL_MAX_DEL
                    AY);
                     }

}
```