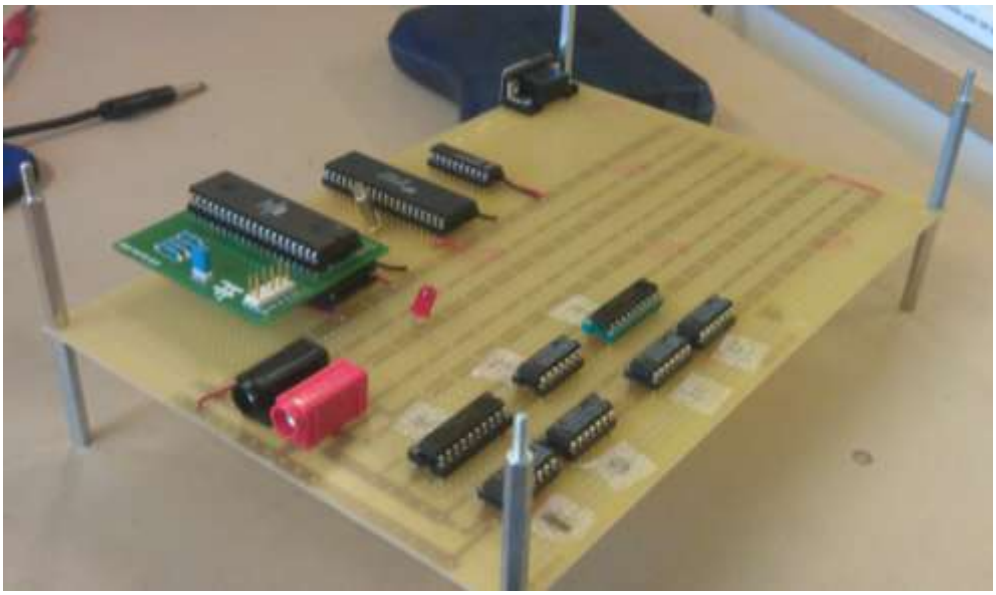
KRYPTERING

23 maj

2012

A short report regarding the difficulties of encrypting traffic between a AVRmega16 and a computer, using a DUART as intermediate. The rapport states that it is not impossible for three novices in the field to accomplish such a task, even though it is more laboursome than it seems. It also contains a crash-course in cryptology and the theories covering that field.

Vilken jävla duart?



Fredrik Leifland, Fredrik Skeppstedt, Peter Dah
I-09

I-09, Fredrik Leifland 900807-0956 fredrik.leifland@gmail.com
I-09, Fredrik Skeppstedt 891123-1952 f.skeppsted@gmail.com
I-09, Peter Dahl 890414-4030 peterdahl_@hotmail.com

Innehållsförteckning

I-09, Fredrik Leifland 900807-0956 fredrik.leifland@gmail.com
I-09, Fredrik Skeppstedt 891123-1952 f.skeppsted@gmail.com
I-09, Peter Dahl 890414-4030 peterdahl_@hotmail.com

I. Inledning

A. Syfte

Att designa och bygga en krets som är kapabel att kryptera signal med 8-bitars säkerhet samt därtill hörande avkrypteringsprogram till en dator. Insignalen skall kunna vara både hårdprogrammerad såväl som inläst.

Detta då det känns som en relevant uppgift i ett samhälle där kryptering blir en allt mer intim del av att bevara integriteten på Internet, framförallt då avlyssningslagarna blir fler och begränsningarna till deras applicerande färre. Författarna är naturligtvis medvetna om att deras projekt inte har någon praktisk applikation, ens efter mycket vidareutveckling, dock är de angelägna om de teoretiska grunder på vilka integritetsfrämjande åtgärder vilar.

B. Problemformulering

För att uppfylla syftet som är formulerat ovan ställdes dessa krav på systemet:

1. Under en cykeltid vara tillsynes slumpmässig i sitt förvrängande av bitar.
2. Ha en cykeltid, period, på 8-bitar.
3. Vara byggt på ett sådant sätt att man skulle kunna skala upp storleken på chiffret vid intresse.
4. Innehålla gränssnitt för mottagning av flera signaler.
5. Kommunicera med en dator via en seriell port.

C. Avgränsningar

Initialt var planen att implementera kryptering av fingeravtryck för vidare kommunikation till dator. Projektet lades upp för detta och därför utelämnades bland annat konstruktion av en fingeravtrycksläsare, en uppgift som ansågs vara alldeles för avancerad får studenternas ingångskunskaper. Så småningom visade det sig dock vara alldeles för svårt att lösa programmeringen av processorns kommunikation till datornvilket medförde att fingeravtrycksläsaren fick stryka på foten.

D. Disposition

Denna rapport är författad i tre segment; inledning, projektet samt utvärdering. Detta för att ge läsaren en tydlig och lättläst text. Huvudfokus kommer naturligtvis att ligga på mittendelen då det är den som dokumenterar och beskriver vad som har genomförts.

I-09, Fredrik Leifland 900807-0956 fredrik.leifland@gmail.com
I-09, Fredrik Skeppstedt 891123-1952 f.skeppsted@gmail.com
I-09, Peter Dahl 890414-4030 peterdahl_@hotmail.com

II. Teori

A. Krypteringsteori

Vår krypteringsmetod bygger på Grain-kryptering, vilket innebär att man först krypterar signalen linjärt genom ett linjärt återkopplat skiftregister (Linear Feedback Shift Register, LFSR), sedan skickas den krypterade signalen vidare till ett Non-Linear Feedback Shift Register (NLFSR) för att ytterligare kastas om. Slutligen kombineras utsignalen från dessa två i en logisk port innan det skickas tillbaka till sändaren.

Nycklen till detta krypto är att man vet ursprungsläget när man påbörjar krypteringen av meddelandet, det vill säga att man vet vad som står var i de två registerna. Utöver detta behöver man även veta med vilken matetmatisk funktion man skiftar bitarna internt i registren. Kombinationen av detta gör att man kan lätt och smidigt läsa återskapa de tillstånd som fanns i registren vid dekrypteringstillfället varpå man får ut meddelandet i klartext.

1. LFSR

Ett LFSR baseras på att man på vissa platser i registret lyssnar av innehållet i just den cellen, detta kombineras sedan med vad som finns i cellen före den man ska skriva i genom addition modulo två. På detta sätt byts tecknen i registret ut linjärt och funktionen i sig är enkel att plotta. Dock så räcker inte denna kryptering då det är tämligen lätt att knäcka den även med begränsad datorkraft varför man skickar för vidare signalströmmen in i ett NLFSR.

2. NLFSR

Ett NLFSR är ett olinjärt LFSR, det vill säga att man inte med samma säkerhet kan förutsätta hur en bit kommer att ändras, om den kommer att ändras. Detta då NLFSR:et likt LFSR:et lyssnar på tidigare celler i registret men istället för att addera innehållet så multiplicerar det modulo två, ofta i längre cykler med en tre-fyra celler inblandade som input till en enda. Det är detta som gör NLFSR-krypteringen säkrare, då det är mer avancerade funktioner som måste knäckas för att få innehållet läsbart.

3. Cykeltider

Alla kryptons säkerhet beror på kryptots cykeltid, det vill säga hur många klockningar, bitar, innan man är tillbaka på ruta ett. Detta gäller eftersom man för varje period lättare kan bygga upp ett mönster över hur chiffret beter sig och därmed också kan dekryptera signalen lättare.

B. I hårdvara

Precis som det står ovan så baseras signalkryptering på att det förefaller slumpmässigt om det är en etta eller en nolla som skickas näst. I ett hårdvarukrypto, vilket är vad vi ämnar bygga, innebär det att man måste lagra en ström av ettor och nollor i ett register. Detta görs genom så kallade D-vippor som antingen har värde ett (en laddning på 5V) eller värde noll (en laddning på 0V). För att göra strömmen till synes slumpmässig, enligt ovanstående teknik, krävs det att man via logiska portar (främst XOR- och AND-portar) återför innehållet i de olika D-vipporna till varandra. Det vill säga stör den ursprungliga ordningen på bitarna.

Figur 1 – Kryptot

I-09, Fredrik Leifland 900807-0956 fredrik.leifland@gmail.com
I-09, Fredrik Skeppstedt 891123-1952 f.skeppsted@gmail.com
I-09, Peter Dahl 890414-4030 peterdahl_@hotmail.com

III. Genomförande

A. Design

För att kunna bygga något krävs det en utförlig plan på hur komponenterna skall sitta ihop samt var de skall placeras på kretsen. Detta designades i PowerLogic och ganska snart kunde komponenterna inhandlas och konstruktionen startas. Det som var svårast att designa var i särklass chifferkomponenten, som består av två D-vippor och ett antal logiska grindar av olika sort. För att kontrollera att det gjordes någorlunda enligt rätt principer konsulterades kryptologiexperter på institutionen. Dessa rekommenderade ett krypto i storleksordningen 128 bitar men vi fick av kostnads och komplexitetsskäl rådet av handledaren att nöja oss med 8 bitars säkerhet, uppbyggt av ett 8 bitars LFSR och ett 8 bitars NLFSR enligt principerna presenterade tidigare. Nedan finns algoritmerna som designades för de två delarna av chiffret.

$$((Q_8 + \text{Input}) \oplus Q_5) \oplus (Q_1 \oplus Q_2)$$

Figur 2 – Algoritmen för LFSR-chiffret

$$(((\text{LFSR}_7 \oplus (Q_1 * Q_6)) \oplus ((Q_5 * Q_7) \oplus (Q_3 \oplus Q_4)) \oplus Q_8) \oplus \text{LFSR}_1)$$

Figur 3 – Algoritmen för NLFSR-chiffret

Grundtanken var som tidigare sagt att låta en fingeravtrycksläsare ta ett fingeravtryck och via en bit-ström sedan kryptera detta genom vårt chiffer, skicka den krypterade strömmen till en dator för att där koda av den i ett mjukvaruprogram.

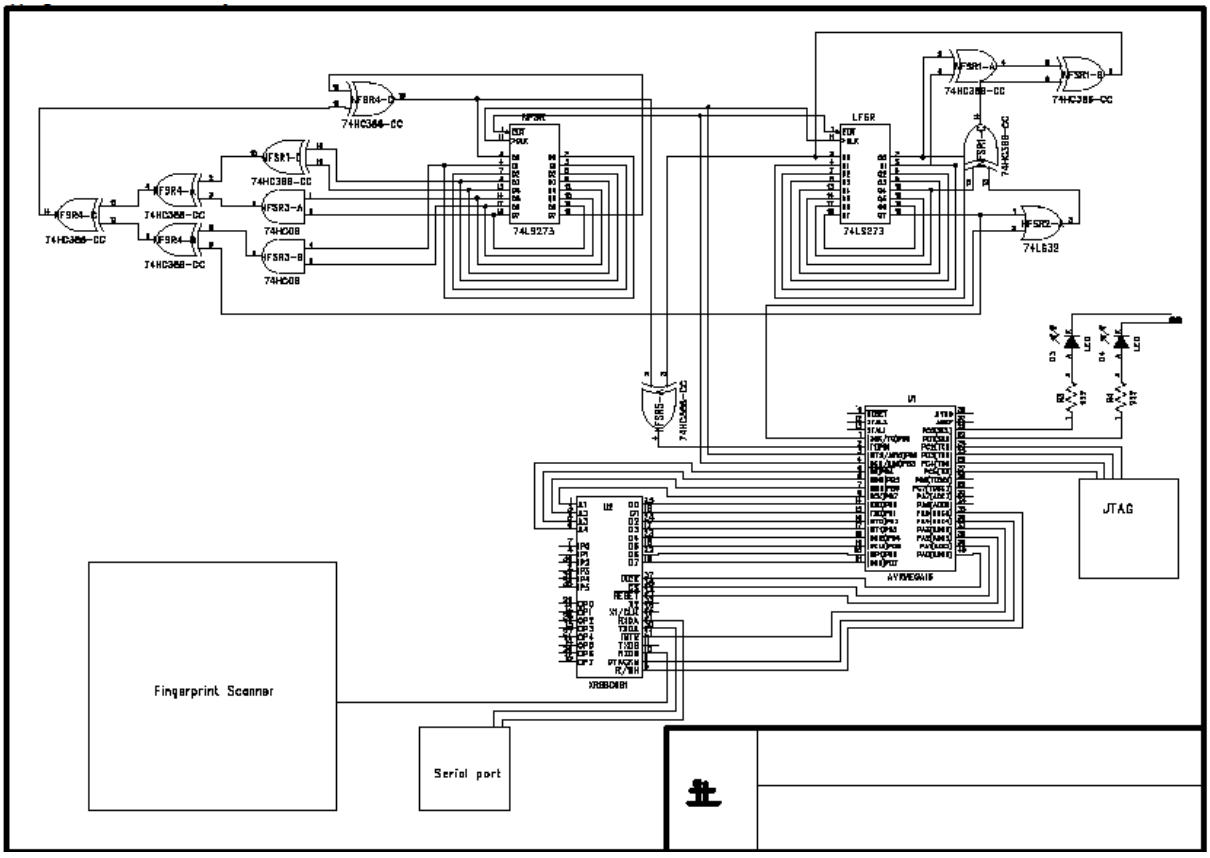
1. Dual Asynchronous Receiver/Transmitter – DUART

Då projektet hade fler inkällor till processorn än vad processorn hade lediga pinnar/portar övervägdes först en lösning med en mer komplicerad processor. Denna förkastades dock till fördel för en lösning där man lät en DUART ta hand om den extra kommunikationen och sedan vidare förmedla den till processorn. Kommunikationen skulle inte ske simultant varför detta ansågs vara en mer kostnadseffektiv och smidigare lösning. Dock visade sig DUART:en vara så pass komplicerad att förstå och konfiguera varför projektet fick återföras till designfasen där fingeravtrycksläsaren slopades, detta då det inte ansågs rimligt att implementera även den givet tidsramarna.

2. Komponentlista och kopplingsschema

Processor:	1 st	ATmega16
Kommunikationskrets:	1 st	X68C681 – Dual Asynchronus Receiver/Transmitter (DUART)
Feedbackkrets:	1 st	Röd LED-diod
Klocka:	1 st	16MHz kristall.
<u>Kryptot</u>		
Chifferminnen:	2 st	74LS273- oktal D-vippa med klocka och nollställning
Logiska portar		
AND port:	1 st	74HC08N – kvadrupel positiv AND-port
XOR port:	3 st	74HC86N – kvadrupel XOR-port
OR port:	1 st	74HC32N – kvadrupel positiv OR-port

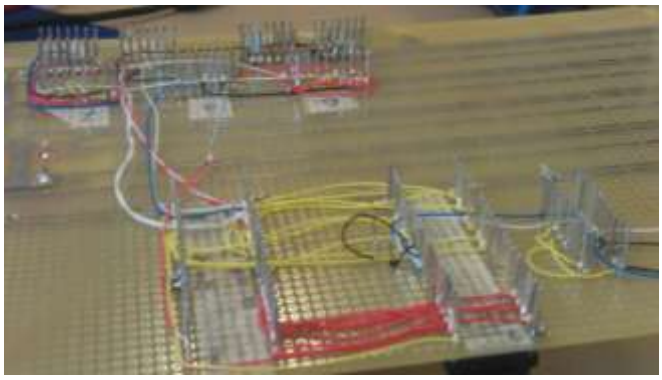
I-09, Fredrik Leifland 900807-0956 fredrik.leifland@gmail.com
I-09, Fredrik Skeppstedt 891123-1952 f.skeppsted@gmail.com
I-09, Peter Dahl 890414-4030 peterdahl_@hotmail.com



Figur 4 – Kopplingsschema, ursprunglig version

B. Bygget

Så snart som komponenterna var på plats och det sista finliret på designen var avklarad påbörjades den fysiska konstruktionen av kretsen. Detta inleddes med att samtliga komponenter placerades ut varpå strömtillförsel och jord löddes fast vid för ändamålet avsedda ben. Så snart detta var klart påbörjades virningen av LSFR-delen av kryptot, detta följdes i sin tur av virningen av NFSR-delen. Så snart kryptot var klart så fästes in-/utsignalsladdarna på rätt ställen men lämnades lösa. Så snart kryptot var färdigvirat påbörjades processen att sammanlänka vår processor med DUART:en samt vidare därifrån till serieporten, via en transformator. När detta var klart återstod bara att koppla in en LED-diod och klockan innan allt var färdigt att programmeras. I detta skede bestämdes att vänta med att koppla in fingeravtryckläsaren då det ansågs viktigare att sätta upp en baseline med grundprogrammering så att kommunikationen med datorn via DUART:en kunde upprättas.



Figur 5 – Virningen

C. C-Programmeringen

Programmeringen till processorn sköttes via AVRstudio som genom en Jtag-modul anslöt till den samme. Kodskrivandet var uppbyggt så att författarna tidigt skulle få en uppfattning om att rätt saker

skedde i kretsen., därav behovet av en LED-diod som skulle kunna blinka vid när en ny rad kod exekverades. Systematiskt skrevs programmet upp från grunden för att täcka samtliga händelseförlopp i processorn och kretsen, det var främst vi konfigurationen av DUART:en som problemen uppstod.

1. DUART:en

Då en DUART är en komplicerad komponent, med lika många pinnar som AVR mega16, innebar det att det var svårt för författarna att felsöka den när signalerna inte gick fram. Tidigt insåg kodproducenten att ett av de stora problemen var kretsens Baud-rate som var alldeles för låg, vilket insågs via diagnosticering med hjälp av ett oscillioskop.

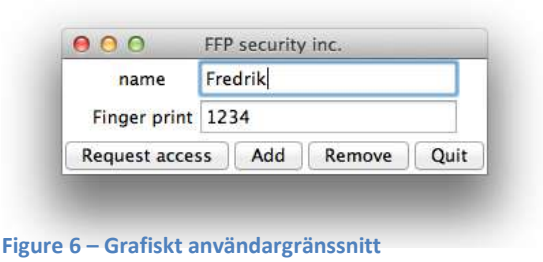


Figure 6 – Grafiskt användargränssnitt

D. Datorprogrammet

Datorklientens uppgift har varit att hantera en databas över godkända fingeravtryck och att dekryptera de fingeravtryck som skickas till datorn från AVR:en för autentisering. Input skulle ha tagits direkt från seriell port, men detta omprövades p.g.a. dess komplexitet och input till programmet tas nu genom dialogruta. Funktionen för kryptering och dekryptering finns skriven, men är inte ihopkopplad med användargränssnittet eftersom vi inte heller kunde få det att fungera p.g.a. att AVR:en, av oklar anledning, vägrar gå in i en if-sats i vårt C-program på den. Klienten är skriven i Python.

IV. Resultat

Signalerna som skickades in i kryptot krypterades enligt nedanstående algoritm, vilket gjorde det rimligt svårt att knäcka koden. Även om detta i sak inte testatss så är de teoretiska säkerheten tämligen begränsad, främst på grund av den korta cykeltiden. Dock kan man konstatera att signalerna krypterades tillfredställande med denna metod, då bitarna i signalen uppträder till synes slumpvis. När denna bit-ström sedan introducerades i den tillhörande programvaran för dekryptering så återskapades det ursprungliga meddelandet till fullo.

$$(((LFSR_8 + \text{Input}) \oplus LFSR_5) \oplus (LFSR_1 \oplus LFSR_2)) \oplus (LFSR_7 \oplus (NLFSR_1 * NLFSR_6)) \oplus ((NLFSR_5 * NLFSR_7) \oplus (NLFSR_3 \oplus NLFSR_4)) \oplus NLFSR_8) \oplus LFSR_1)$$

Figur 7 – Den kompletta krypteringsalgoritmen

V. Slutsats och utvärdering

A. Slutsats

Det är avsevärt mer tidskrävande att sätta sig in i en så pass komplicerad komponent som en DUART än vad som kan synas. Dock skall inte slutprodukten föraktas då den levererar avsedd funktionalitet.

1. Ambitionerna

Det ska av författarna villigt erkännas att de tog sig vatten över huvudet när de designade grunden för detta system då de inte var inbegripna i DUART:ens komplexitet. Detta gjorde att de hade den ambitiösa viljan att dels konfiguera ovan nämnda komponent utöver att reverse-engineer:a en redan färdigbygg och konfiguerad fingeravtrycksläsare

B. Utvärdering

Projektet har gett en god förståelse för hur en dator kommunicerar med en krets via en seriell port, detta är något som delas generellt av samtliga inblandade i projektet. Vad som däremot skiljer sig är inom vilket område som fokus har lagst för den enskilda individen, på C-programmeringen, på den medföljande mjukvaran eller på kretsens logiska uppbyggnad. Detta kan anses vara både en styrka och en svaghet då det otvetydigt har gjort arbetet effektivare på många plan även om kunskapsintegrationen varit lika fullständig. Överlag är samtliga dock överens om att de har fått en god grund inför framtida möten med både elektroingenjörer och eventuella digitala projekt.

VI. Referenser

Bertil Lindvall, Handledare och Forskningsingenjör
Martin Hell, Konstruktör av Grain-kryptot, Universitetslektor och Teknisk Doktor
Thomas Johansson, Konstruktör av Grain-kryptot och Professor

VII. Appendix

A. Källkod

```
#include <avr/io.h>
#define F_CPU 1000000UL  // 1 MHz
#include <util/delay.h>

void set_pin(char port, char pin, char state);
unsigned short int a;
void read(unsigned short int addressing, int
reset)
{
   /* Read to Register */
   if( reset == 1) {
      set_pin('A', PA2, 1);
      set_pin('A', PA2, 0);
      set_pin('A', PA2, 1);
   }

   PORTB = addressing; // addressinG
      DDRD = 0b00000000;
   set_pin('A', PA5, 0);
   set_pin('A', PA5, 1);
   set_pin('A', PA1, 0);
}

void write(unsigned short int  addressing,
unsigned short int  databus, int reset)
{

   // Write to the DUART shizzzle

   if( reset == 1) {
      set_pin('A', PA2, 1);
      set_pin('A', PA2, 0);
      set_pin('A', PA2, 1);
   }

   set_pin('A', PA5, 1);
   set_pin('A', PA1, 1);

   PORTB = addressing; // addressinG
   PORTD = databus; // databus

   set_pin('A', PA5, 0);
   set_pin('A', PA1, 0);

   set_pin('A', PA1, 1);
   set_pin('A', PA5, 1);

}

void tx_uart(char val)
{

   set_pin('A', PA5, 1);
   set_pin('A', PA1, 1);

   PORTB = 0b11000000; // addressinG
   PORTD = val; // databus

   set_pin('A', PA5, 0);
   set_pin('A', PA1, 0);

   set_pin('A', PA1, 1);
   set_pin('A', PA5, 1);
}

unsigned short int clock_Crypto()
{
   unsigned short int slask;

     set_pin('B', PB2, 0);
   set_pin('B', PB2, 1);

   slask=PINB&0b00000010;
   if(slask != 0x00){
      return 0b00000001;
   } else {
      return 0b00000000;
   }
}

void tx_uart_encrypt()
{
   unsigned short int final;
   int dum = 0;
```

```
  unsigned short int array[] = {0b10000000,
0b01000000, 0b00100000, 0b00010000,
0b00001000, 0b00000100,
0b0000010,0b00000001};
  unsigned short int zero = 0b00000000;
//0b01000110

  for(int k = 0; k <= 7; k++) {
    a = clock_Crypto();
    if(a == 0b00000000) {
      final = final | zero;
      dum++;
      }
    else {
      final = final | array[k];
      dum--;
      }
  }


  set_pin('A', PA5, 1);
  set_pin('A', PA1, 1);

  PORTB = 0b11000000; // addressinG
  PORTD = final;

  set_pin('A', PA5, 0);
  set_pin('A', PA1, 0);

  set_pin('A', PA1, 1);
  set_pin('A', PA5, 1);
}

int rx_uart()
{
  char val;

  set_pin('A', PA1, 1);
  set_pin('A', PA5, 1);

  PORTB = 0b11000000; // addressinG
  DDRD = 0b00000000;

  set_pin('A', PA5, 0);
  set_pin('A', PA5, 1);
  set_pin('A', PA1, 0);

  val = PORTD;
  DDRD = 0b11111111;
  return val;
}

void init_Crypto()
{
  int key[] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0};
  int key1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  set_pin('B', PB3, 0);
  set_pin('B', PB3, 1);

  for( int k = 0; k <= 15; k++ ) {
    if(key[k] == 1) {
      set_pin('B', PB0, 1);

    } else {
      set_pin('B', PB0, 0);
    }
    set_pin('B', PB2, 0);
    set_pin('B', PB2, 1);
  }
}


void init_duart()
{
  //Setting the Data Direction Registers
  DDRA = 0b11111111;
  DDRB = 0b11111101;
  DDRC = 0b00000111;
  DDRD = 0b11111111;

  // Write to MR1A
  write(0b00000000, 0b00010011, 1);
//0b00010010

  // Write to MR2A
  write(0b00000000, 0b00000111, 0);
//0b00010111
```

```
  // Write to CRA
  write(0b01000000, 0x0A, 0); //0b00000101

  write(0b01000000, 0x80, 0);

  // Write to ACR
  write(0b00100000, 0b10110000, 0);
//0b11000000

    // Write to CSRA
  write(0b10000000, 0xBB, 0); //

  // Write to CRA
  write(0b01000000, 0x05, 0); //0b00000101
}

void main(void)
{

  init_Crypto();
  set_pin('C', PC1, 1);
   init_duart();

  while(1)
  {
  set_pin('C', PC1, 0);
    clock_Crypto();
    //_delay_ms(10);
    tx_uart(0b01000110);
    //_delay_ms(10);
    tx_uart_encrypt();
    _delay_ms(10);
    //tx_uart('D');
    _delay_ms(10);
    //tx_uart('R');
    _delay_ms(10);
    //tx_uart('I');
    _delay_ms(10);
    //tx_uart('K');
    _delay_ms(1000);
    //tx_uart(' ');

    set_pin('C', PC1, 1);
  }
  return;
}

void set_pin(char port, char pin, char state){
  char set = 1 << pin;
  if(port == 'A'){
    set &= PORTA;
    if(set && !state){ //ändra från 1 -> 0
      PORTA ^= set;
    }
    if(set == 0 && state){ //ändra från 0 -> 1
      set = 1 << pin;
      PORTA ^= set;
    }
  }
  else if(port == 'B'){
    set &= PORTB;
    if(set && !state){ //ändra från 1 -> 0
      PORTB ^= set;
    }
    if(set == 0 && state){ //ändra från 0 -> 1
      set = 1 << pin;
      PORTB ^= set;
    }
  }
  else if(port == 'C'){
    set &= PORTC;
    if(set && !state){ //ändra från 1 -> 0
      PORTC ^= set;
    }
    if(set == 0 && state){ //ändra från 0 -> 1
      set = 1 << pin;
      PORTC ^= set;
    }
  }
  else if(port == 'D'){
    set &= PORTD;
    if(set && !state){ //ändra från 1 -> 0
      PORTD ^= set;
    }
    if(set == 0 && state){ //ändra från 0 -> 1
      set = 1 << pin;
      PORTD ^= set;
    }
  }
}
```