# Формулировка задания

Получение практических навыков по организации двунаправленного взаимодействия между микроконтроллером и персональным компьютером. Знакомство с криптосистемой RSA и криптографическими протоколами Диффи-Хеллмана и Эль-Гамаля.

Доп. Задание (вариант 3):

Криптосистема RSA. На МК должен выполняться сбор статистики: количество обращений (шифрование, расшифрование), количество обращений с отсутствующим в памяти открытым ключом, суммарный объём зашифрованных и расшифрованных файлов. Вся статистика должна записываться в EEPROM и передаваться в программу на ПК для отображения.

# Схема лабораторной установки

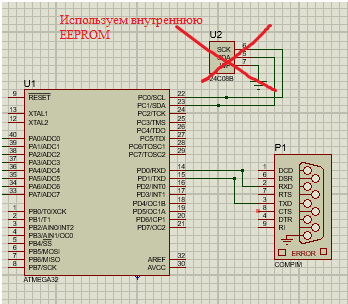


Рис. 1 «Схема лабораторной установки»

# Форматы пересылаемых данных (в оба направления)

ПЭВМ и МК взаимодействуют с помощью интерфейса USART, отсылая друг другу пакеты с постоянным размером BUF\_SIZE. На МК данные передаются следующим образом: первый байт – это код функции, которая находится в таблице функций - обработчиков, второй байт – количество аргументов данной функции. После этого передаются аргументы, записанные сразу после первых двух байт. В ответ МК заполняет первые 2 байта ответного пакета следующим из двух значений: 11 – успех, 00 – неудача. Если операция была выполнена успешно, то после первых двух байт записывается полезная нагрузка: структуры с запрошенными значениями или массив целочисленных значений. В противном случае за байтами записывается причина ошибки в виде текстового сообщения.

# Описание интерфейса программы для ПЭВМ

ПЭВМ написана на языке программирования С. Программа предоставляет пользователю набор функций для удобной работы с криптосистемой:

* Add <n> <e> <d> - добавить новые параметры RSA
* Del <e> - удалить параметры n e d, с открытым ключом e
* Enc <in.txt> <out.txt> - зашифровать файл
* Dec <in.txt> <out.txt> - расшифровать файл
* Gen – сгенерировать параметры RSA
* Clear – очистить EEPROM
* Keys – показать список доступных открытых ключей
* Set <e> - установить параметры для работы, где e – открытый ключ
* Stats – показать статистику

Программа считывает эти команды с помощью командной строки, затем выполняет операцию и выводит на консоль ответ от МК.

Пример ввода команды из cmd: ASVTcrypt.exe COM4 add 213 45600 4557

# Блок-схема алгоритма работы программы для МК

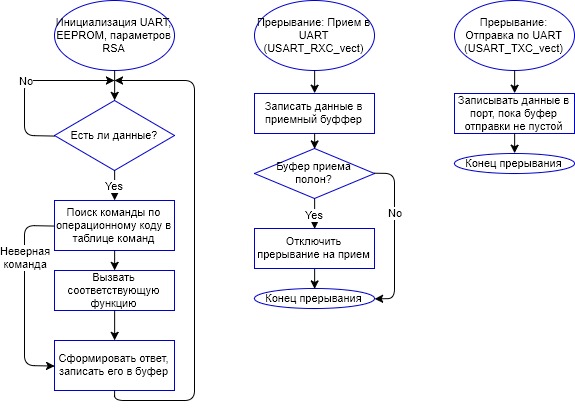


Рис. 2. Блок-схема алгоритма работы программы для МК

# Блок-схема алгоритма работы программы для ПЭВМ

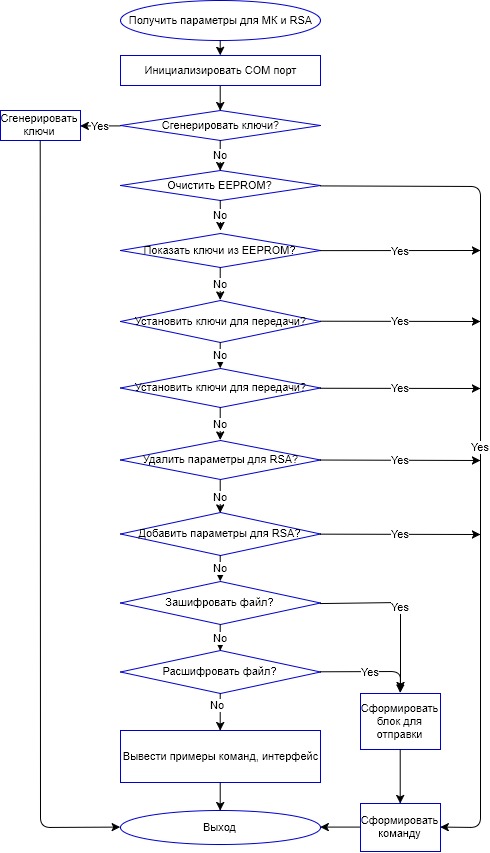


Рис. 3. Блок-схема алгоритма работы программы для ПЭВМ

# Временные диаграммы логических сигналов на портах МК (фрагмент)

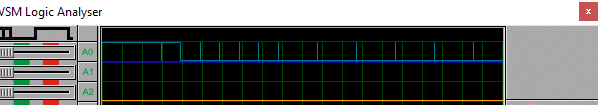


Рис. 4 «Передача пакета с ПЭВМ на МК»

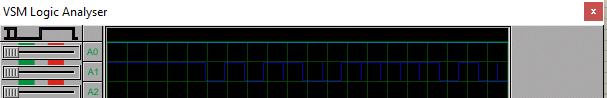


Рис. 5 «Передача пакета с МК на ПЭВМ»

# Результаты работы

В результате работы был создан прототип криптосистемы. Программа на МК написана на языке программирования С. Программный код разделен на 4 региона: реализация базовых функцийEEPROM, реализация базовых функций USART, протокол передачи данных по USART и интерфейс криптосистемы. Рассмотрим поподробнее интерфейс криптосистемы.

Основные функции:

* CryptInit – инициализация криптосистемы. Сначала инициализируется память EEPROM. Так как в памяти могут храниться данные с прошлого сеанса, то в структуру NED, содержащую открытый и закрытый ключи, было введено поле magic – случайное значение, помечающее запись как принадлежащую криптосистеме. Далее отводится память под статистику и в глобальной памяти сохраняется её адрес. Остальная память отводится под хранение ключей.
* CryptPrepare – функция, которая перед началом шифрования/расшифрования проверяет, установлены ли параметры, а также обновляет статистику. После её успешного завершения вызываются функции EncryptTextBlock и DecryptTextBlock.
* EncryptTextBlock – зашифровать блок текста с применением rsa
* DecryptTextBlock - расшифровать блок текста с применением rsa
* AddCryptParameters – сохраняет параметры rsa в eeprom. Если места нет, то возвращает ошибку
* DelCryptParameters - удаляет параметры rsaизeeprom.
* ShowCryptStats – передает ПЭВМ статистику
* CryptShowKeys - передает ПЭВМ список доступных ключей
* CryptClearEEPROM – очищает EEPROM, заново её инициализируя

Получив пакет с данными, функция mainпроверяет корректность данных, затем вызывает одну из вышеописанных функций. После этого отправляет ответное текстовое сообщение ПЭВМ.

# Выводы по лабораторной работе

Криптосистема является полезным аппаратным средством для шифрования и расшифровки данных. Благодаря тому, что параметры RSA хранятся в EEPROM, эта система становится удобной в регулярном использовании.

# Приложение 1 Комментированный листинг программы для МК на языке Си

#define F\_CPU 8000000ul

//#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <string.h>

#pragma region EEPROM\_RW

// eeprom: 0 .. 1023 -> 1024 bytes

#define EEPROM\_SZ 1024

uint32\_t gEEPROM;

uint32\_t gEndEEPROM;

void EEPROM\_init()

{

gEEPROM = (uint32\_t)EEARH;

gEEPROM <<= 8;

gEEPROM |= EEARL;

gEndEEPROM = gEEPROM + EEPROM\_SZ;

}

void EEPROM\_write\_byte(uint32\_t uiAddress, uint8\_t ucData)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEWE));

/\* Set up address and data registers \*/

EEAR = uiAddress;

EEDR = ucData;

/\* Write logical one to EEMWE \*/

EECR |= (1<<EEMWE);

/\* Start eeprom write by setting EEWE \*/

EECR |= (1<<EEWE);

}

void EEPROM\_write(uint32\_t uiAddress, void \*data, uint8\_t nBytes)

{

uint8\_t i;

uint8\_t \*ucdata = data;

for (i = 0; i < nBytes; i++)

{

EEPROM\_write\_byte(uiAddress + i, ucdata[i]);

}

}

uint8\_t EEPROM\_read\_byte(uint32\_t uiAddress)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEWE));

/\* Set up address register \*/

EEAR = uiAddress;

/\* Start eeprom read by writing EERE \*/

EECR |= (1<<EERE);

/\* Return data from data register \*/

return EEDR;

}

void EEPROM\_read(uint32\_t uiAddress, void \*data, uint8\_t nBytes)

{

uint8\_t i;

uint8\_t \*ucdata = data;

for (i = 0; i < nBytes; i++)

{

ucdata[i] = EEPROM\_read\_byte(uiAddress + i);

}

}

#pragma endregion EEPROM\_RW

#pragma region UsartBasicFunctions

void USART\_Init()

{

/\* Set baud rate. Freq = 8.0 MHz. 9600 <-> UBRR = 51\*/

unsigned int baud = 51;

UBRRH = (uint8\_t)(baud>>8);

UBRRL = (uint8\_t)baud;

/\* Enable receiver and transmitter \*/

UCSRB = (1<<RXEN)|(1<<TXEN);

// enable interrupt on data reception and transmition

UCSRB |= (1<<RXCIE);

UCSRB |= (1<<TXCIE);

/\* Set frame format: 8data, 1 stop bit \*/

UCSRC = (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);

}

void USART\_Transmit( uint8\_t data )

{

/\* Wait for empty transmit buffer \*/

while ( !( UCSRA & (1<<UDRE)) );

/\* Put data into buffer, sends the data \*/

UDR = data;

}

unsigned int USART\_Receive( void )

{

uint8\_t status, resh, resl;

/\* Wait for data to be received \*/

while ( !(UCSRA & (1<<RXC)) );

/\* Get status and 9th bit, then data \*/

/\* from buffer \*/

status = UCSRA;

resh = UCSRB;

resl = UDR;

/\*

Bit 4 – FE: Frame Error

Bit 3 – DOR: Data OverRun

Bit 2 – PE: Parity Error

\*/

/\* If error, return -1 \*/

if (status & ((1<<FE)|(1<<DOR)|(1<<PE)))

return -1;

/\* Filter the 9th bit, then return \*/

resh = (resh >> 1) & 0x01;

return ((resh << 8) | resl);

return UDR;

}

#pragma endregion UsartBasicFunctions

#pragma region UsartSendRecieve

// recv bytes

uint8\_t gBytesRecv = 0; // amount of bytes which were received

uint8\_t gDataReady = 0; // if it's one then user can read received data

// recv buffer size

#define BUF\_SZ 254

char gRecvBuf[BUF\_SZ];

uint8\_t USART\_data\_ready()

{

return gDataReady != 0;

}

void USART\_read\_data(char buffer[BUF\_SZ])

{

// interrupts disabled

memcpy(buffer, gRecvBuf, BUF\_SZ);

memset(gRecvBuf, 0, BUF\_SZ);

gBytesRecv = 0;

gDataReady = 0;

sei();

}

// bit RCX is 1 when byte receiving is finished

ISR(USART\_RXC\_vect)

{

gRecvBuf[gBytesRecv] = USART\_Receive();

if (++gBytesRecv == BUF\_SZ)

{

// disable interrupts and read data

gDataReady = 1;

cli();

}

}

uint8\_t gBytesSent = 0; // amount of bytes which were sent

char gSendBuf[BUF\_SZ];

ISR(USART\_TXC\_vect)

{

if (gBytesSent < BUF\_SZ)

USART\_Transmit(gSendBuf[gBytesSent++]);

else

gBytesSent = 0;

}

void USART\_write\_data(char buf[BUF\_SZ])

{

memset(gSendBuf, 0, BUF\_SZ);

memcpy(gSendBuf, buf, BUF\_SZ);

// start transmition

gBytesSent++;

USART\_Transmit(gSendBuf[0]);

}

#pragma endregion UsartSendRecieve

#pragma region CryptoInterface

// value for slot

#define EEPROM\_magic 0xFAFEFDFC

// command = operation code + number of args + arguments

uint8\_t g\_nSlots = 0; // available slots for recording

uint32\_t gStatistics\_addr; // address of statistics storage

// statictics

typedef struct stats

{

uint32\_t magic;

uint32\_t nEncrypt;

uint32\_t nDecrypt;

uint32\_t nRequestNoOpenKey;

uint32\_t nEncryptedBytes;

uint32\_t nDecryptedBytes;

} STATS;

// RSA

typedef struct nde

{

uint32\_t magic;

uint32\_t n; // module

uint32\_t e; // open key

uint32\_t d; // secret key

} NDE;

// current RSA parameters

NDE gCurNde = { 0 };

#define NDE\_SZ sizeof(NDE)

#define STATS\_SZ sizeof(STATS)

void CryptInit()

{

// write stats to the end of EEPROM

gStatistics\_addr = gEndEEPROM - STATS\_SZ - sizeof(uint32\_t) - sizeof(uint32\_t);

STATS st;

EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

// if no magic -> no records were done before, so init statistics

if (st.magic != EEPROM\_magic)

{

memset(&st, 0, STATS\_SZ);

st.magic = EEPROM\_magic;

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

}

// check eeprom memory for previous records

g\_nSlots = (EEPROM\_SZ - STATS\_SZ) / NDE\_SZ;

uint8\_t i;

for (i = 0; i < g\_nSlots; i++)

{

uint32\_t addr = gEEPROM + i \* NDE\_SZ;

NDE Nde;

EEPROM\_read(addr, &Nde, NDE\_SZ);

// this is not a NDE record, so fill it with zeroes

if (Nde.magic != EEPROM\_magic)

{

memset(&Nde, 0, NDE\_SZ);

EEPROM\_write(addr, &Nde, NDE\_SZ);

}

}

}

// command struct with argc

struct Com

{

void (\*handler)(char[BUF\_SZ], char[BUF\_SZ]);

uint8\_t argc;

};

// find if condition is true

uint32\_t find\_if(NDE x, uint8\_t (\*func\_if)(NDE slot\_elem, NDE myelem))

{

NDE temp;

int i;

for (i = 0; i < g\_nSlots; i++)

{

uint32\_t addr = gEEPROM + i \* NDE\_SZ;

EEPROM\_read(addr, &temp, NDE\_SZ);

if (func\_if(temp, x))

return addr;

}

return (uint32\_t)-1;

}

uint8\_t e\_equal(NDE temp, NDE x)

{

return (temp.magic == EEPROM\_magic && temp.e == x.e);

}

uint8\_t slot\_empty(NDE temp, NDE x)

{

return temp.magic != EEPROM\_magic;

}

uint32\_t powmod(uint32\_t base, uint32\_t exp, uint32\_t mod)

{

uint32\_t res = 1;

while (exp != 0)

{

if ((exp & 1) != 0)

{

res = (1u \* res \* base) % mod;

}

base = (1u \* base \* base) % mod;

exp >>= 1;

}

return res;

}

void CryptPrepare(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

if (gCurNde.magic != EEPROM\_magic)

strcpy(out\_str, "00Encryption parameters n,e,d are not set");

else

{

uint8\_t opcode = str[2] - '0';

STATS st;

EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

st.nRequestNoOpenKey++;

if (opcode == 0)

{

st.nEncrypt++;

strcpy(out\_str, "11Encryption will be started now...");

}

else

{

st.nDecrypt++;

strcpy(out\_str, "11Decryption will be started now...");

}

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

}

}

// only half of str buffer will be used

void EncryptTextBlock(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

// encrypt: uint8\_t -> uint16\_t

uint8\_t offs = sizeof(uint8\_t) + sizeof(uint8\_t) + sizeof(uint32\_t);

memcpy(out\_str, str, offs);

uint32\_t nIters = \*(uint32\_t\*)(str + 2);

uint8\_t \*buf = (uint8\_t \*)(str + offs);

uint16\_t \*out\_buf = (uint16\_t \*)(out\_str + offs);

uint8\_t i;

// ecrypt

for (i = 0; i < nIters; i++)

out\_buf[i] = powmod(buf[i], gCurNde.e, gCurNde.n);

// refresh stats

STATS st;

EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

st.nEncryptedBytes += nIters;

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

}

// only half of out\_str buffer will be used

void DecryptTextBlock(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

// decrypt: uint16\_t -> uint8\_t

uint8\_t offs = sizeof(uint8\_t) + sizeof(uint8\_t) + sizeof(uint32\_t);

memcpy(out\_str, str, offs);

uint32\_t nIters = \*(uint32\_t\*)(str + 2);

uint16\_t \*buf = (uint16\_t \*)(str + offs);

uint8\_t \*out\_buf = (uint8\_t \*)(out\_str + offs);

uint8\_t i;

// decrypt

for (i = 0; i < nIters; i++)

out\_buf[i] = powmod(buf[i], gCurNde.d, gCurNde.n);

// refresh stats

STATS st;

EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

st.nDecryptedBytes += nIters;

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

}

// add new client's RSA n e d

void AddCryptParameters(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

NDE to\_add = \*(NDE\*)(str + 2);

to\_add.magic = EEPROM\_magic;

uint32\_t empty\_slot\_addr = find\_if(to\_add, slot\_empty);

if (empty\_slot\_addr != -1)

{

EEPROM\_write(empty\_slot\_addr, &to\_add, NDE\_SZ);

strcpy(out\_str, "11Parameters are added to container");

}

else

strcpy(out\_str, "00No free space in EEPROM");

}

// find and del slot where e = client's 'e'

void DelCryptParameters(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

NDE to\_del = \*(NDE\*)(str + 2);

uint32\_t del\_nde\_addr = find\_if(to\_del, e\_equal);

if (del\_nde\_addr != (uint32\_t)-1)

{

NDE empty = { 0 };

EEPROM\_write(del\_nde\_addr, &empty, NDE\_SZ);

strcpy(out\_str, "11Parameters are deleted from container");

}

else

strcpy(out\_str, "00Parameters are not found in container");

}

void inc\_stat\_nook()

{

// refresh stats

STATS st;

EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

st.nRequestNoOpenKey++;

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

}

// send statistics to client

void ShowCryptStats(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

out\_str[0] = out\_str[1] = '1';

EEPROM\_read(gStatistics\_addr, out\_str + 2, STATS\_SZ);

inc\_stat\_nook();

}

// set current RSA parameters

void SetCryptParams(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

NDE x = \*(NDE\*)(str + 2);

uint32\_t secret\_addr = find\_if(x, e\_equal);

// secret key exists

if (secret\_addr != -1)

{

EEPROM\_read(secret\_addr, &gCurNde, NDE\_SZ);

strcpy(out\_str, "11Encryption parameters are set");

}

else

strcpy(out\_str, "00Encryption parameters weren't found");

}

// send to client all keys found in EEPROM

void CryptShowKeys(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

uint8\_t i, k = 0;

NDE Nde;

uint32\_t \*out = (uint32\_t \*)(out\_str);

for (i = 0; i < g\_nSlots; i++)

{

uint32\_t addr = gEEPROM + i \* NDE\_SZ;

EEPROM\_read(addr, &Nde, NDE\_SZ);

if (Nde.magic == EEPROM\_magic)

out[k++] = Nde.e;

}

inc\_stat\_nook();

}

void CryptClearEEPROM(char str[BUF\_SZ], char out\_str[BUF\_SZ])

{

uint8\_t i;

NDE Nde = { 0 };

for (i = 0; i < g\_nSlots; i++)

{

uint32\_t addr = gEEPROM + i \* NDE\_SZ;

EEPROM\_write(addr, &Nde, NDE\_SZ);

}

// clear statistics

STATS st = {EEPROM\_magic, 0, 0, 0};

EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

strcpy(out\_str, "11EEPROM was cleared");

}

struct Com CommandTable[] =

{

{ EncryptTextBlock, '2' },

{ DecryptTextBlock, '2' },

{ AddCryptParameters, '3' },

{ DelCryptParameters, '1' },

{ SetCryptParams, '1' },

{ ShowCryptStats, '0' },

{ CryptPrepare, '1' },

{ CryptShowKeys, '0' },

{ CryptClearEEPROM, '0' },

};

#pragma endregion CryptoInterface

int main(void)

{

DDRA = 0xFF;

EEPROM\_init();

PORTA = 0x01;

CryptInit();

PORTA = 0x02;

USART\_Init();

PORTA = 0xFF;

\_delay\_ms(500);

PORTA = 0;

sei();

while (1)

{

// wait until data arrives

if (USART\_data\_ready())

{

char readBuf[BUF\_SZ] = { 0 }, writeBuf[BUF\_SZ] = { 0 };

USART\_read\_data(readBuf);

uint8\_t opcode = readBuf[0];

// find command by operation code in command table

if (opcode > '8' || opcode < '0')

strcpy(writeBuf, "00Operation code is invalid");

else

{

opcode -= '0';

if (CommandTable[opcode].argc != readBuf[1])

strcpy(writeBuf, "00This function has another count of arguments");

else

CommandTable[opcode].handler(readBuf, writeBuf);

}

USART\_write\_data(writeBuf);

}

else

\_delay\_ms(200);

}

}

# Приложение 2 Комментированный листинг программы для МК на языке ассемблера

--- ../../../../crt1/gcrt1.S ---------------------------------------------------

00000043 RCALL PC+0x0755 Relative call subroutine

00000044 JMP 0x00000861 Jump

00000046 RJMP PC-0x0046 Relative jump

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

331: {

00000047 PUSH R10 Push register on stack

00000048 PUSH R11 Push register on stack

00000049 PUSH R12 Push register on stack

0000004A PUSH R13 Push register on stack

0000004B PUSH R14 Push register on stack

0000004C PUSH R15 Push register on stack

0000004D PUSH R16 Push register on stack

0000004E PUSH R17 Push register on stack

0000004F PUSH R28 Push register on stack

00000050 PUSH R29 Push register on stack

00000051 IN R28,0x3D In from I/O location

00000052 IN R29,0x3E In from I/O location

00000053 SBIW R28,0x10 Subtract immediate from word

00000054 IN R0,0x3F In from I/O location

00000055 CLI Global Interrupt Disable

00000056 OUT 0x3E,R29 Out to I/O location

00000057 OUT 0x3F,R0 Out to I/O location

00000058 OUT 0x3D,R28 Out to I/O location

00000059 STD Y+1,R10 Store indirect with displacement

0000005A STD Y+2,R11 Store indirect with displacement

0000005B STD Y+3,R12 Store indirect with displacement

0000005C STD Y+4,R13 Store indirect with displacement

0000005D STD Y+9,R18 Store indirect with displacement

0000005E STD Y+10,R19 Store indirect with displacement

0000005F STD Y+11,R20 Store indirect with displacement

00000060 STD Y+12,R21 Store indirect with displacement

332: return (temp.magic == EEPROM\_magic && temp.e == x.e);

00000061 LDD R24,Y+1 Load indirect with displacement

00000062 LDD R25,Y+2 Load indirect with displacement

00000063 LDD R26,Y+3 Load indirect with displacement

00000064 LDD R27,Y+4 Load indirect with displacement

00000065 CPI R24,0xFC Compare with immediate

00000066 SBCI R25,0xFD Subtract immediate with carry

00000067 SBCI R26,0xFE Subtract immediate with carry

00000068 SBCI R27,0xFA Subtract immediate with carry

00000069 BRNE PC+0x11 Branch if not equal

--- Нет исходного файла --------------------------------------------------------

0000006A LDI R24,0x01 Load immediate

0000006B LDD R16,Y+9 Load indirect with displacement

0000006C LDD R17,Y+10 Load indirect with displacement

0000006D LDD R18,Y+11 Load indirect with displacement

0000006E LDD R19,Y+12 Load indirect with displacement

0000006F LDD R20,Y+37 Load indirect with displacement

00000070 LDD R21,Y+38 Load indirect with displacement

00000071 LDD R22,Y+39 Load indirect with displacement

00000072 LDD R23,Y+40 Load indirect with displacement

00000073 CP R16,R20 Compare

00000074 CPC R17,R21 Compare with carry

00000075 CPC R18,R22 Compare with carry

00000076 CPC R19,R23 Compare with carry

00000077 BREQ PC+0x04 Branch if equal

00000078 LDI R24,0x00 Load immediate

00000079 RJMP PC+0x0002 Relative jump

0000007A LDI R24,0x00 Load immediate

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

333: }

0000007B ADIW R28,0x10 Add immediate to word

0000007C IN R0,0x3F In from I/O location

0000007D CLI Global Interrupt Disable

0000007E OUT 0x3E,R29 Out to I/O location

0000007F OUT 0x3F,R0 Out to I/O location

00000080 OUT 0x3D,R28 Out to I/O location

00000081 POP R29 Pop register from stack

00000082 POP R28 Pop register from stack

00000083 POP R17 Pop register from stack

00000084 POP R16 Pop register from stack

00000085 POP R15 Pop register from stack

00000086 POP R14 Pop register from stack

00000087 POP R13 Pop register from stack

00000088 POP R12 Pop register from stack

00000089 POP R11 Pop register from stack

0000008A POP R10 Pop register from stack

0000008B RET Subroutine return

336: {

0000008C PUSH R10 Push register on stack

0000008D PUSH R11 Push register on stack

0000008E PUSH R12 Push register on stack

0000008F PUSH R13 Push register on stack

00000090 PUSH R14 Push register on stack

00000091 PUSH R15 Push register on stack

00000092 PUSH R16 Push register on stack

00000093 PUSH R17 Push register on stack

00000094 PUSH R28 Push register on stack

00000095 PUSH R29 Push register on stack

00000096 IN R28,0x3D In from I/O location

00000097 IN R29,0x3E In from I/O location

00000098 SBIW R28,0x10 Subtract immediate from word

00000099 IN R0,0x3F In from I/O location

0000009A CLI Global Interrupt Disable

0000009B OUT 0x3E,R29 Out to I/O location

0000009C OUT 0x3F,R0 Out to I/O location

0000009D OUT 0x3D,R28 Out to I/O location

0000009E STD Y+1,R10 Store indirect with displacement

0000009F STD Y+2,R11 Store indirect with displacement

000000A0 STD Y+3,R12 Store indirect with displacement

000000A1 STD Y+4,R13 Store indirect with displacement

337: return temp.magic != EEPROM\_magic;

000000A2 LDI R24,0x01 Load immediate

000000A3 LDD R20,Y+1 Load indirect with displacement

000000A4 LDD R21,Y+2 Load indirect with displacement

000000A5 LDD R22,Y+3 Load indirect with displacement

000000A6 LDD R23,Y+4 Load indirect with displacement

000000A7 CPI R20,0xFC Compare with immediate

000000A8 SBCI R21,0xFD Subtract immediate with carry

000000A9 SBCI R22,0xFE Subtract immediate with carry

000000AA SBCI R23,0xFA Subtract immediate with carry

000000AB BRNE PC+0x02 Branch if not equal

000000AC LDI R24,0x00 Load immediate

338: }

000000AD ADIW R28,0x10 Add immediate to word

000000AE IN R0,0x3F In from I/O location

000000AF CLI Global Interrupt Disable

000000B0 OUT 0x3E,R29 Out to I/O location

000000B1 OUT 0x3F,R0 Out to I/O location

000000B2 OUT 0x3D,R28 Out to I/O location

000000B3 POP R29 Pop register from stack

000000B4 POP R28 Pop register from stack

000000B5 POP R17 Pop register from stack

000000B6 POP R16 Pop register from stack

000000B7 POP R15 Pop register from stack

000000B8 POP R14 Pop register from stack

000000B9 POP R13 Pop register from stack

000000BA POP R12 Pop register from stack

000000BB POP R11 Pop register from stack

000000BC POP R10 Pop register from stack

000000BD RET Subroutine return

116: gEEPROM = (uint32\_t)EEARH;

000000BE IN R24,0x1F In from I/O location

117: gEEPROM <<= 8;

000000BF LDI R25,0x00 Load immediate

000000C0 LDI R26,0x00 Load immediate

000000C1 LDI R27,0x00 Load immediate

000000C2 MOV R27,R26 Copy register

000000C3 MOV R26,R25 Copy register

000000C4 MOV R25,R24 Copy register

000000C5 CLR R24 Clear Register

000000C6 STS 0x0432,R24 Store direct to data space

000000C8 STS 0x0433,R25 Store direct to data space

000000CA STS 0x0434,R26 Store direct to data space

000000CC STS 0x0435,R27 Store direct to data space

118: gEEPROM |= EEARL;

000000CE IN R18,0x1E In from I/O location

000000CF OR R24,R18 Logical OR

000000D0 STS 0x0432,R24 Store direct to data space

000000D2 STS 0x0433,R25 Store direct to data space

000000D4 STS 0x0434,R26 Store direct to data space

000000D6 STS 0x0435,R27 Store direct to data space

119: gEndEEPROM = gEEPROM + EEPROM\_SZ;

000000D8 SUBI R25,0xFC Subtract immediate

000000D9 SBCI R26,0xFF Subtract immediate with carry

000000DA SBCI R27,0xFF Subtract immediate with carry

000000DB STS 0x043A,R24 Store direct to data space

000000DD STS 0x043B,R25 Store direct to data space

000000DF STS 0x043C,R26 Store direct to data space

000000E1 STS 0x043D,R27 Store direct to data space

000000E3 RET Subroutine return

125: while(EECR & (1<<EEWE));

000000E4 SBIC 0x1C,1 Skip if bit in I/O register cleared

000000E5 RJMP PC-0x0001 Relative jump

128: EEAR = uiAddress;

000000E6 OUT 0x1F,R23 Out to I/O location

000000E7 OUT 0x1E,R22 Out to I/O location

129: EEDR = ucData;

000000E8 OUT 0x1D,R20 Out to I/O location

132: EECR |= (1<<EEMWE);

000000E9 SBI 0x1C,2 Set bit in I/O register

135: EECR |= (1<<EEWE);

000000EA SBI 0x1C,1 Set bit in I/O register

000000EB RET Subroutine return

139: {

000000EC PUSH R12 Push register on stack

000000ED PUSH R13 Push register on stack

000000EE PUSH R14 Push register on stack

000000EF PUSH R15 Push register on stack

000000F0 PUSH R16 Push register on stack

000000F1 PUSH R17 Push register on stack

000000F2 PUSH R28 Push register on stack

000000F3 PUSH R29 Push register on stack

143: for (i = 0; i < nBytes; i++)

000000F4 TST R18 Test for Zero or Minus

000000F5 BREQ PC+0x19 Branch if equal

000000F6 MOVW R28,R20 Copy register pair

000000F7 SUBI R18,0x01 Subtract immediate

000000F8 MOV R16,R18 Copy register

000000F9 LDI R17,0x00 Load immediate

000000FA SUBI R16,0xFF Subtract immediate

000000FB SBCI R17,0xFF Subtract immediate with carry

000000FC ADD R16,R20 Add without carry

000000FD ADC R17,R21 Add with carry

000000FE MOV R12,R22 Copy register

000000FF MOV R13,R23 Copy register

00000100 MOV R14,R24 Copy register

00000101 MOV R15,R25 Copy register

145: EEPROM\_write\_byte(uiAddress + i, ucdata[i]);

00000102 LD R20,Y+ Load indirect and postincrement

00000103 MOVW R24,R14 Copy register pair

00000104 MOVW R22,R12 Copy register pair

00000105 RCALL PC-0x0021 Relative call subroutine

00000106 SER R24 Set Register

00000107 SUB R12,R24 Subtract without carry

00000108 SBC R13,R24 Subtract with carry

00000109 SBC R14,R24 Subtract with carry

0000010A SBC R15,R24 Subtract with carry

143: for (i = 0; i < nBytes; i++)

0000010B CP R28,R16 Compare

0000010C CPC R29,R17 Compare with carry

0000010D BRNE PC-0x0B Branch if not equal

147: }

0000010E POP R29 Pop register from stack

0000010F POP R28 Pop register from stack

00000110 POP R17 Pop register from stack

00000111 POP R16 Pop register from stack

00000112 POP R15 Pop register from stack

00000113 POP R14 Pop register from stack

00000114 POP R13 Pop register from stack

00000115 POP R12 Pop register from stack

00000116 RET Subroutine return

521: {

00000117 PUSH R14 Push register on stack

00000118 PUSH R15 Push register on stack

00000119 PUSH R17 Push register on stack

0000011A PUSH R28 Push register on stack

0000011B PUSH R29 Push register on stack

0000011C IN R28,0x3D In from I/O location

0000011D IN R29,0x3E In from I/O location

0000011E SBIW R28,0x22 Subtract immediate from word

0000011F IN R0,0x3F In from I/O location

00000120 CLI Global Interrupt Disable

00000121 OUT 0x3E,R29 Out to I/O location

00000122 OUT 0x3F,R0 Out to I/O location

00000123 OUT 0x3D,R28 Out to I/O location

00000124 MOVW R14,R22 Copy register pair

523: NDE Nde = { 0 };

00000125 MOVW R30,R28 Copy register pair

00000126 ADIW R30,0x01 Add immediate to word

00000127 LDI R24,0x10 Load immediate

00000128 MOVW R26,R30 Copy register pair

00000129 ST X+,R1 Store indirect and postincrement

0000012A DEC R24 Decrement

0000012B BRNE PC-0x02 Branch if not equal

524: for (i = 0; i < g\_nSlots; i++)

0000012C LDS R24,0x0232 Load direct from data space

0000012E TST R24 Test for Zero or Minus

0000012F BREQ PC+0x1E Branch if equal

00000130 LDI R17,0x00 Load immediate

527: EEPROM\_write(addr, &Nde, NDE\_SZ);

00000131 LDI R27,0x10 Load immediate

00000132 MUL R17,R27 Multiply unsigned

00000133 MOVW R18,R0 Copy register pair

00000134 CLR R1 Clear Register

00000135 LDS R24,0x0432 Load direct from data space

00000137 LDS R25,0x0433 Load direct from data space

00000139 LDS R26,0x0434 Load direct from data space

0000013B LDS R27,0x0435 Load direct from data space

0000013D MOVW R22,R24 Copy register pair

0000013E MOVW R24,R26 Copy register pair

0000013F ADD R22,R18 Add without carry

00000140 ADC R23,R19 Add with carry

00000141 ADC R24,R1 Add with carry

00000142 ADC R25,R1 Add with carry

00000143 LDI R18,0x10 Load immediate

00000144 MOVW R20,R28 Copy register pair

00000145 SUBI R20,0xFF Subtract immediate

00000146 SBCI R21,0xFF Subtract immediate with carry

00000147 RCALL PC-0x005B Relative call subroutine

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

524: for (i = 0; i < g\_nSlots; i++)

00000148 SUBI R17,0xFF Subtract immediate

00000149 LDS R24,0x0232 Load direct from data space

0000014B CP R17,R24 Compare

0000014C BRCS PC-0x1B Branch if carry set

531: STATS st = {EEPROM\_magic, 0, 0, 0};

0000014D MOVW R20,R28 Copy register pair

0000014E SUBI R20,0xEF Subtract immediate

0000014F SBCI R21,0xFF Subtract immediate with carry

00000150 LDI R24,0x12 Load immediate

00000151 MOVW R30,R20 Copy register pair

00000152 ST Z+,R1 Store indirect and postincrement

00000153 DEC R24 Decrement

00000154 BRNE PC-0x02 Branch if not equal

00000155 LDI R24,0xFC Load immediate

00000156 LDI R25,0xFD Load immediate

00000157 LDI R26,0xFE Load immediate

00000158 LDI R27,0xFA Load immediate

00000159 STD Y+17,R24 Store indirect with displacement

0000015A STD Y+18,R25 Store indirect with displacement

0000015B STD Y+19,R26 Store indirect with displacement

0000015C STD Y+20,R27 Store indirect with displacement

532: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

0000015D LDS R22,0x0436 Load direct from data space

0000015F LDS R23,0x0437 Load direct from data space

00000161 LDS R24,0x0438 Load direct from data space

00000163 LDS R25,0x0439 Load direct from data space

00000165 LDI R18,0x12 Load immediate

00000166 RCALL PC-0x007A Relative call subroutine

534: strcpy(out\_str, "11EEPROM was cleared");

00000167 LDI R24,0x15 Load immediate

00000168 LDI R30,0x7B Load immediate

00000169 LDI R31,0x00 Load immediate

0000016A MOVW R26,R14 Copy register pair

0000016B LD R0,Z+ Load indirect and postincrement

0000016C ST X+,R0 Store indirect and postincrement

0000016D DEC R24 Decrement

0000016E BRNE PC-0x03 Branch if not equal

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

535: }

0000016F ADIW R28,0x22 Add immediate to word

00000170 IN R0,0x3F In from I/O location

00000171 CLI Global Interrupt Disable

00000172 OUT 0x3E,R29 Out to I/O location

00000173 OUT 0x3F,R0 Out to I/O location

00000174 OUT 0x3D,R28 Out to I/O location

00000175 POP R29 Pop register from stack

00000176 POP R28 Pop register from stack

00000177 POP R17 Pop register from stack

00000178 POP R15 Pop register from stack

00000179 POP R14 Pop register from stack

0000017A RET Subroutine return

152: while(EECR & (1<<EEWE));

0000017B SBIC 0x1C,1 Skip if bit in I/O register cleared

0000017C RJMP PC-0x0001 Relative jump

155: EEAR = uiAddress;

0000017D OUT 0x1F,R23 Out to I/O location

0000017E OUT 0x1E,R22 Out to I/O location

158: EECR |= (1<<EERE);

0000017F SBI 0x1C,0 Set bit in I/O register

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

161: return EEDR;

00000180 IN R24,0x1D In from I/O location

162: }

00000181 RET Subroutine return

165: {

00000182 PUSH R12 Push register on stack

00000183 PUSH R13 Push register on stack

00000184 PUSH R14 Push register on stack

00000185 PUSH R15 Push register on stack

00000186 PUSH R16 Push register on stack

00000187 PUSH R17 Push register on stack

00000188 PUSH R28 Push register on stack

00000189 PUSH R29 Push register on stack

168: for (i = 0; i < nBytes; i++)

0000018A TST R18 Test for Zero or Minus

0000018B BREQ PC+0x19 Branch if equal

0000018C MOVW R28,R20 Copy register pair

0000018D SUBI R18,0x01 Subtract immediate

0000018E MOV R16,R18 Copy register

0000018F LDI R17,0x00 Load immediate

00000190 SUBI R16,0xFF Subtract immediate

00000191 SBCI R17,0xFF Subtract immediate with carry

00000192 ADD R16,R20 Add without carry

00000193 ADC R17,R21 Add with carry

00000194 MOV R12,R22 Copy register

00000195 MOV R13,R23 Copy register

00000196 MOV R14,R24 Copy register

00000197 MOV R15,R25 Copy register

170: ucdata[i] = EEPROM\_read\_byte(uiAddress + i);

00000198 MOVW R24,R14 Copy register pair

00000199 MOVW R22,R12 Copy register pair

0000019A RCALL PC-0x001F Relative call subroutine

0000019B ST Y+,R24 Store indirect and postincrement

0000019C SER R24 Set Register

0000019D SUB R12,R24 Subtract without carry

0000019E SBC R13,R24 Subtract with carry

0000019F SBC R14,R24 Subtract with carry

000001A0 SBC R15,R24 Subtract with carry

168: for (i = 0; i < nBytes; i++)

000001A1 CP R28,R16 Compare

000001A2 CPC R29,R17 Compare with carry

000001A3 BRNE PC-0x0B Branch if not equal

172: }

000001A4 POP R29 Pop register from stack

000001A5 POP R28 Pop register from stack

000001A6 POP R17 Pop register from stack

000001A7 POP R16 Pop register from stack

000001A8 POP R15 Pop register from stack

000001A9 POP R14 Pop register from stack

000001AA POP R13 Pop register from stack

000001AB POP R12 Pop register from stack

000001AC RET Subroutine return

359: {

000001AD PUSH R14 Push register on stack

000001AE PUSH R15 Push register on stack

000001AF PUSH R17 Push register on stack

000001B0 PUSH R28 Push register on stack

000001B1 PUSH R29 Push register on stack

000001B2 IN R28,0x3D In from I/O location

000001B3 IN R29,0x3E In from I/O location

000001B4 SBIW R28,0x12 Subtract immediate from word

000001B5 IN R0,0x3F In from I/O location

000001B6 CLI Global Interrupt Disable

000001B7 OUT 0x3E,R29 Out to I/O location

000001B8 OUT 0x3F,R0 Out to I/O location

000001B9 OUT 0x3D,R28 Out to I/O location

000001BA MOVW R26,R22 Copy register pair

360: if (gCurNde.magic != EEPROM\_magic)

000001BB LDS R20,0x0222 Load direct from data space

000001BD LDS R21,0x0223 Load direct from data space

000001BF LDS R22,0x0224 Load direct from data space

000001C1 LDS R23,0x0225 Load direct from data space

000001C3 CPI R20,0xFC Compare with immediate

000001C4 SBCI R21,0xFD Subtract immediate with carry

000001C5 SBCI R22,0xFE Subtract immediate with carry

000001C6 SBCI R23,0xFA Subtract immediate with carry

000001C7 BREQ PC+0x09 Branch if equal

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

361: strcpy(out\_str, "00Encryption parameters n,e,d are not set");

000001C8 LDI R24,0x2A Load immediate

000001C9 LDI R30,0x90 Load immediate

000001CA LDI R31,0x00 Load immediate

000001CB LD R0,Z+ Load indirect and postincrement

000001CC ST X+,R0 Store indirect and postincrement

000001CD DEC R24 Decrement

000001CE BRNE PC-0x03 Branch if not equal

000001CF RJMP PC+0x0040 Relative jump

000001D0 MOVW R14,R26 Copy register pair

000001D1 MOVW R30,R24 Copy register pair

364: uint8\_t opcode = str[2] - '0';

000001D2 LDD R17,Z+2 Load indirect with displacement

367: EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

000001D3 LDS R22,0x0436 Load direct from data space

000001D5 LDS R23,0x0437 Load direct from data space

000001D7 LDS R24,0x0438 Load direct from data space

000001D9 LDS R25,0x0439 Load direct from data space

000001DB LDI R18,0x12 Load immediate

000001DC MOVW R20,R28 Copy register pair

000001DD SUBI R20,0xFF Subtract immediate

000001DE SBCI R21,0xFF Subtract immediate with carry

000001DF RCALL PC-0x005D Relative call subroutine

368: st.nRequestNoOpenKey++;

000001E0 LDD R24,Y+9 Load indirect with displacement

000001E1 LDD R25,Y+10 Load indirect with displacement

000001E2 ADIW R24,0x01 Add immediate to word

000001E3 STD Y+10,R25 Store indirect with displacement

000001E4 STD Y+9,R24 Store indirect with displacement

370: if (opcode == 0)

000001E5 CPI R17,0x30 Compare with immediate

000001E6 BRNE PC+0x0F Branch if not equal

372: st.nEncrypt++;

000001E7 LDD R24,Y+5 Load indirect with displacement

000001E8 LDD R25,Y+6 Load indirect with displacement

000001E9 ADIW R24,0x01 Add immediate to word

000001EA STD Y+6,R25 Store indirect with displacement

000001EB STD Y+5,R24 Store indirect with displacement

373: strcpy(out\_str, "11Encryption will be started now...");

000001EC LDI R24,0x24 Load immediate

000001ED LDI R30,0xBA Load immediate

000001EE LDI R31,0x00 Load immediate

000001EF MOVW R26,R14 Copy register pair

000001F0 LD R0,Z+ Load indirect and postincrement

000001F1 ST X+,R0 Store indirect and postincrement

000001F2 DEC R24 Decrement

000001F3 BRNE PC-0x03 Branch if not equal

000001F4 RJMP PC+0x000E Relative jump

378: st.nDecrypt++;

000001F5 LDD R24,Y+7 Load indirect with displacement

000001F6 LDD R25,Y+8 Load indirect with displacement

000001F7 ADIW R24,0x01 Add immediate to word

000001F8 STD Y+8,R25 Store indirect with displacement

000001F9 STD Y+7,R24 Store indirect with displacement

379: strcpy(out\_str, "11Decryption will be started now...");

000001FA LDI R24,0x24 Load immediate

000001FB LDI R30,0xDE Load immediate

000001FC LDI R31,0x00 Load immediate

000001FD MOVW R26,R14 Copy register pair

000001FE LD R0,Z+ Load indirect and postincrement

000001FF ST X+,R0 Store indirect and postincrement

00000200 DEC R24 Decrement

00000201 BRNE PC-0x03 Branch if not equal

382: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

00000202 LDS R22,0x0436 Load direct from data space

00000204 LDS R23,0x0437 Load direct from data space

00000206 LDS R24,0x0438 Load direct from data space

00000208 LDS R25,0x0439 Load direct from data space

0000020A LDI R18,0x12 Load immediate

0000020B MOVW R20,R28 Copy register pair

0000020C SUBI R20,0xFF Subtract immediate

0000020D SBCI R21,0xFF Subtract immediate with carry

0000020E RCALL PC-0x0122 Relative call subroutine

384: }

0000020F ADIW R28,0x12 Add immediate to word

00000210 IN R0,0x3F In from I/O location

00000211 CLI Global Interrupt Disable

00000212 OUT 0x3E,R29 Out to I/O location

00000213 OUT 0x3F,R0 Out to I/O location

00000214 OUT 0x3D,R28 Out to I/O location

00000215 POP R29 Pop register from stack

00000216 POP R28 Pop register from stack

00000217 POP R17 Pop register from stack

00000218 POP R15 Pop register from stack

00000219 POP R14 Pop register from stack

0000021A RET Subroutine return

179: UBRRH = (uint8\_t)(baud>>8);

0000021B OUT 0x20,R1 Out to I/O location

180: UBRRL = (uint8\_t)baud;

0000021C LDI R24,0x33 Load immediate

0000021D OUT 0x09,R24 Out to I/O location

183: UCSRB = (1<<RXEN)|(1<<TXEN);

0000021E LDI R24,0x18 Load immediate

0000021F OUT 0x0A,R24 Out to I/O location

186: UCSRB |= (1<<RXCIE);

00000220 SBI 0x0A,7 Set bit in I/O register

187: UCSRB |= (1<<TXCIE);

00000221 SBI 0x0A,6 Set bit in I/O register

190: UCSRC = (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);

00000222 LDI R24,0x86 Load immediate

00000223 OUT 0x20,R24 Out to I/O location

00000224 RET Subroutine return

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

196: while ( !( UCSRA & (1<<UDRE)) );

00000225 SBIS 0x0B,5 Skip if bit in I/O register set

00000226 RJMP PC-0x0001 Relative jump

199: UDR = data;

00000227 OUT 0x0C,R24 Out to I/O location

00000228 RET Subroutine return

207: while ( !(UCSRA & (1<<RXC)) );

00000229 SBIS 0x0B,7 Skip if bit in I/O register set

0000022A RJMP PC-0x0001 Relative jump

212: status = UCSRA;

0000022B IN R25,0x0B In from I/O location

213: resh = UCSRB;

0000022C IN R24,0x0A In from I/O location

214: resl = UDR;

0000022D IN R20,0x0C In from I/O location

223: if (status & ((1<<FE)|(1<<DOR)|(1<<PE)))

0000022E ANDI R25,0x1C Logical AND with immediate

0000022F BRNE PC+0x08 Branch if not equal

228: return ((resh << 8) | resl);

00000230 LSR R24 Logical shift right

00000231 ANDI R24,0x01 Logical AND with immediate

00000232 MOV R19,R24 Copy register

00000233 LDI R18,0x00 Load immediate

00000234 MOVW R24,R18 Copy register pair

00000235 OR R24,R20 Logical OR

00000236 RET Subroutine return

224: return -1;

00000237 SER R24 Set Register

00000238 SER R25 Set Register

231: }

00000239 RET Subroutine return

239: {

0000023A MOVW R26,R24 Copy register pair

241: memcpy(buffer, gRecvBuf, BUF\_SZ);

0000023B LDI R25,0xFE Load immediate

0000023C LDI R30,0x36 Load immediate

0000023D LDI R31,0x02 Load immediate

0000023E LD R0,Z+ Load indirect and postincrement

0000023F ST X+,R0 Store indirect and postincrement

00000240 DEC R25 Decrement

00000241 BRNE PC-0x03 Branch if not equal

242: memset(gRecvBuf, 0, BUF\_SZ);

00000242 LDI R24,0xFE Load immediate

00000243 LDI R30,0x36 Load immediate

00000244 LDI R31,0x02 Load immediate

00000245 MOVW R26,R30 Copy register pair

00000246 ST X+,R1 Store indirect and postincrement

00000247 DEC R24 Decrement

00000248 BRNE PC-0x02 Branch if not equal

243: gBytesRecv = 0;

00000249 STS 0x0235,R1 Store direct to data space

244: gDataReady = 0;

0000024B STS 0x0234,R1 Store direct to data space

245: sei();

0000024D SEI Global Interrupt Enable

0000024E RET Subroutine return

250: {

0000024F PUSH R1 Push register on stack

00000250 PUSH R0 Push register on stack

00000251 IN R0,0x3F In from I/O location

00000252 PUSH R0 Push register on stack

00000253 CLR R1 Clear Register

00000254 PUSH R18 Push register on stack

00000255 PUSH R19 Push register on stack

00000256 PUSH R20 Push register on stack

00000257 PUSH R21 Push register on stack

00000258 PUSH R22 Push register on stack

00000259 PUSH R23 Push register on stack

0000025A PUSH R24 Push register on stack

0000025B PUSH R25 Push register on stack

0000025C PUSH R26 Push register on stack

0000025D PUSH R27 Push register on stack

0000025E PUSH R28 Push register on stack

0000025F PUSH R29 Push register on stack

00000260 PUSH R30 Push register on stack

00000261 PUSH R31 Push register on stack

251: gRecvBuf[gBytesRecv] = USART\_Receive();

00000262 LDS R28,0x0235 Load direct from data space

00000264 LDI R29,0x00 Load immediate

00000265 RCALL PC-0x003C Relative call subroutine

00000266 SUBI R28,0xCA Subtract immediate

00000267 SBCI R29,0xFD Subtract immediate with carry

00000268 STD Y+0,R24 Store indirect with displacement

253: if (++gBytesRecv == BUF\_SZ)

00000269 LDS R24,0x0235 Load direct from data space

0000026B SUBI R24,0xFF Subtract immediate

0000026C STS 0x0235,R24 Store direct to data space

0000026E CPI R24,0xFE Compare with immediate

0000026F BRNE PC+0x05 Branch if not equal

256: gDataReady = 1;

00000270 LDI R24,0x01 Load immediate

00000271 STS 0x0234,R24 Store direct to data space

257: cli();

00000273 CLI Global Interrupt Disable

259: }

00000274 POP R31 Pop register from stack

00000275 POP R30 Pop register from stack

00000276 POP R29 Pop register from stack

00000277 POP R28 Pop register from stack

00000278 POP R27 Pop register from stack

00000279 POP R26 Pop register from stack

0000027A POP R25 Pop register from stack

0000027B POP R24 Pop register from stack

0000027C POP R23 Pop register from stack

0000027D POP R22 Pop register from stack

0000027E POP R21 Pop register from stack

0000027F POP R20 Pop register from stack

00000280 POP R19 Pop register from stack

00000281 POP R18 Pop register from stack

00000282 POP R0 Pop register from stack

00000283 OUT 0x3F,R0 Out to I/O location

00000284 POP R0 Pop register from stack

00000285 POP R1 Pop register from stack

00000286 RETI Interrupt return

262: {

00000287 PUSH R1 Push register on stack

00000288 PUSH R0 Push register on stack

00000289 IN R0,0x3F In from I/O location

0000028A PUSH R0 Push register on stack

0000028B CLR R1 Clear Register

0000028C PUSH R18 Push register on stack

0000028D PUSH R19 Push register on stack

0000028E PUSH R20 Push register on stack

0000028F PUSH R21 Push register on stack

00000290 PUSH R22 Push register on stack

00000291 PUSH R23 Push register on stack

00000292 PUSH R24 Push register on stack

00000293 PUSH R25 Push register on stack

00000294 PUSH R26 Push register on stack

00000295 PUSH R27 Push register on stack

00000296 PUSH R30 Push register on stack

00000297 PUSH R31 Push register on stack

263: if (gBytesSent < BUF\_SZ)

00000298 LDS R30,0x0233 Load direct from data space

0000029A CPI R30,0xFE Compare with immediate

0000029B BRCC PC+0x0B Branch if carry cleared

264: USART\_Transmit(gSendBuf[gBytesSent++]);

0000029C LDI R24,0x01 Load immediate

0000029D ADD R24,R30 Add without carry

0000029E STS 0x0233,R24 Store direct to data space

000002A0 LDI R31,0x00 Load immediate

000002A1 SUBI R30,0xCC Subtract immediate

000002A2 SBCI R31,0xFC Subtract immediate with carry

000002A3 LDD R24,Z+0 Load indirect with displacement

000002A4 RCALL PC-0x007F Relative call subroutine

000002A5 RJMP PC+0x0003 Relative jump

266: gBytesSent = 0;

000002A6 STS 0x0233,R1 Store direct to data space

267: }

000002A8 POP R31 Pop register from stack

000002A9 POP R30 Pop register from stack

000002AA POP R27 Pop register from stack

000002AB POP R26 Pop register from stack

000002AC POP R25 Pop register from stack

000002AD POP R24 Pop register from stack

000002AE POP R23 Pop register from stack

000002AF POP R22 Pop register from stack

000002B0 POP R21 Pop register from stack

000002B1 POP R20 Pop register from stack

000002B2 POP R19 Pop register from stack

000002B3 POP R18 Pop register from stack

000002B4 POP R0 Pop register from stack

000002B5 OUT 0x3F,R0 Out to I/O location

000002B6 POP R0 Pop register from stack

000002B7 POP R1 Pop register from stack

000002B8 RETI Interrupt return

270: {

000002B9 PUSH R28 Push register on stack

000002BA PUSH R29 Push register on stack

000002BB MOVW R30,R24 Copy register pair

271: memset(gSendBuf, 0, BUF\_SZ);

000002BC LDI R28,0x34 Load immediate

000002BD LDI R29,0x03 Load immediate

000002BE LDI R24,0xFE Load immediate

000002BF MOVW R26,R28 Copy register pair

000002C0 ST X+,R1 Store indirect and postincrement

000002C1 DEC R24 Decrement

000002C2 BRNE PC-0x02 Branch if not equal

272: memcpy(gSendBuf, buf, BUF\_SZ);

000002C3 LDI R25,0xFE Load immediate

000002C4 MOVW R26,R28 Copy register pair

000002C5 LD R0,Z+ Load indirect and postincrement

000002C6 ST X+,R0 Store indirect and postincrement

000002C7 DEC R25 Decrement

000002C8 BRNE PC-0x03 Branch if not equal

275: gBytesSent++;

000002C9 LDS R24,0x0233 Load direct from data space

000002CB SUBI R24,0xFF Subtract immediate

000002CC STS 0x0233,R24 Store direct to data space

276: USART\_Transmit(gSendBuf[0]);

000002CE LDD R24,Y+0 Load indirect with displacement

000002CF RCALL PC-0x00AA Relative call subroutine

277: }

000002D0 POP R29 Pop register from stack

000002D1 POP R28 Pop register from stack

000002D2 RET Subroutine return

280: {

000002D3 PUSH R11 Push register on stack

000002D4 PUSH R12 Push register on stack

000002D5 PUSH R13 Push register on stack

000002D6 PUSH R14 Push register on stack

000002D7 PUSH R15 Push register on stack

000002D8 PUSH R16 Push register on stack

000002D9 PUSH R17 Push register on stack

000002DA PUSH R28 Push register on stack

000002DB PUSH R29 Push register on stack

000002DC IN R28,0x3D In from I/O location

000002DD IN R29,0x3E In from I/O location

000002DE SBIW R28,0x22 Subtract immediate from word

000002DF IN R0,0x3F In from I/O location

000002E0 CLI Global Interrupt Disable

000002E1 OUT 0x3E,R29 Out to I/O location

000002E2 OUT 0x3F,R0 Out to I/O location

000002E3 OUT 0x3D,R28 Out to I/O location

282: gStatistics\_addr = gEndEEPROM - STATS\_SZ;

000002E4 LDS R24,0x043A Load direct from data space

000002E6 LDS R25,0x043B Load direct from data space

000002E8 LDS R26,0x043C Load direct from data space

000002EA LDS R27,0x043D Load direct from data space

000002EC MOVW R22,R24 Copy register pair

000002ED MOVW R24,R26 Copy register pair

000002EE SUBI R22,0x12 Subtract immediate

000002EF SBC R23,R1 Subtract with carry

000002F0 SBC R24,R1 Subtract with carry

000002F1 SBC R25,R1 Subtract with carry

000002F2 STS 0x0436,R22 Store direct to data space

000002F4 STS 0x0437,R23 Store direct to data space

000002F6 STS 0x0438,R24 Store direct to data space

000002F8 STS 0x0439,R25 Store direct to data space

284: EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

000002FA LDI R18,0x12 Load immediate

000002FB MOVW R20,R28 Copy register pair

000002FC SUBI R20,0xFF Subtract immediate

000002FD SBCI R21,0xFF Subtract immediate with carry

000002FE RCALL PC-0x017C Relative call subroutine

287: if (st.magic != EEPROM\_magic)

000002FF LDD R24,Y+1 Load indirect with displacement

00000300 LDD R25,Y+2 Load indirect with displacement

00000301 LDD R26,Y+3 Load indirect with displacement

00000302 LDD R27,Y+4 Load indirect with displacement

00000303 CPI R24,0xFC Compare with immediate

00000304 SBCI R25,0xFD Subtract immediate with carry

00000305 SBCI R26,0xFE Subtract immediate with carry

00000306 SBCI R27,0xFA Subtract immediate with carry

00000307 BRNE PC+0x02 Branch if not equal

00000308 RJMP PC+0x0050 Relative jump

289: memset(&st, 0, STATS\_SZ);

00000309 MOVW R20,R28 Copy register pair

0000030A SUBI R20,0xFF Subtract immediate

0000030B SBCI R21,0xFF Subtract immediate with carry

0000030C LDI R24,0x12 Load immediate

0000030D MOVW R30,R20 Copy register pair

0000030E ST Z+,R1 Store indirect and postincrement

0000030F DEC R24 Decrement

00000310 BRNE PC-0x02 Branch if not equal

290: st.magic = EEPROM\_magic;

00000311 LDI R24,0xFC Load immediate

00000312 LDI R25,0xFD Load immediate

00000313 LDI R26,0xFE Load immediate

00000314 LDI R27,0xFA Load immediate

00000315 STD Y+1,R24 Store indirect with displacement

00000316 STD Y+2,R25 Store indirect with displacement

00000317 STD Y+3,R26 Store indirect with displacement

00000318 STD Y+4,R27 Store indirect with displacement

291: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

00000319 LDS R22,0x0436 Load direct from data space

0000031B LDS R23,0x0437 Load direct from data space

0000031D LDS R24,0x0438 Load direct from data space

0000031F LDS R25,0x0439 Load direct from data space

00000321 LDI R18,0x12 Load immediate

00000322 RCALL PC-0x0236 Relative call subroutine

295: g\_nSlots = (EEPROM\_SZ - STATS\_SZ) / NDE\_SZ;

00000323 LDI R24,0x3E Load immediate

00000324 STS 0x0232,R24 Store direct to data space

280: {

00000326 MOV R11,R1 Copy register

303: EEPROM\_read(addr, &Nde, NDE\_SZ);

00000327 MOVW R16,R28 Copy register pair

00000328 SUBI R16,0xED Subtract immediate

00000329 SBCI R17,0xFF Subtract immediate with carry

300: uint32\_t addr = gEEPROM + i \* NDE\_SZ;

0000032A LDI R31,0x10 Load immediate

0000032B MUL R11,R31 Multiply unsigned

0000032C MOVW R24,R0 Copy register pair

0000032D CLR R1 Clear Register

0000032E LDS R12,0x0432 Load direct from data space

00000330 LDS R13,0x0433 Load direct from data space

00000332 LDS R14,0x0434 Load direct from data space

00000334 LDS R15,0x0435 Load direct from data space

00000336 ADD R12,R24 Add without carry

00000337 ADC R13,R25 Add with carry

00000338 ADC R14,R1 Add with carry

00000339 ADC R15,R1 Add with carry

303: EEPROM\_read(addr, &Nde, NDE\_SZ);

0000033A LDI R18,0x10 Load immediate

0000033B MOVW R20,R16 Copy register pair

0000033C MOVW R24,R14 Copy register pair

0000033D MOVW R22,R12 Copy register pair

0000033E RCALL PC-0x01BC Relative call subroutine

306: if (Nde.magic != EEPROM\_magic)

0000033F LDD R24,Y+19 Load indirect with displacement

00000340 LDD R25,Y+20 Load indirect with displacement

00000341 LDD R26,Y+21 Load indirect with displacement

00000342 LDD R27,Y+22 Load indirect with displacement

00000343 CPI R24,0xFC Compare with immediate

00000344 SBCI R25,0xFD Subtract immediate with carry

00000345 SBCI R26,0xFE Subtract immediate with carry

00000346 SBCI R27,0xFA Subtract immediate with carry

00000347 BREQ PC+0x0B Branch if equal

308: memset(&Nde, 0, NDE\_SZ);

00000348 LDI R24,0x10 Load immediate

00000349 MOVW R30,R16 Copy register pair

0000034A ST Z+,R1 Store indirect and postincrement

0000034B DEC R24 Decrement

0000034C BRNE PC-0x02 Branch if not equal

309: EEPROM\_write(addr, &Nde, NDE\_SZ);

0000034D LDI R18,0x10 Load immediate

0000034E MOVW R20,R16 Copy register pair

0000034F MOVW R24,R14 Copy register pair

00000350 MOVW R22,R12 Copy register pair

00000351 RCALL PC-0x0265 Relative call subroutine

298: for (i = 0; i < g\_nSlots; i++)

00000352 INC R11 Increment

00000353 LDS R24,0x0232 Load direct from data space

00000355 CP R11,R24 Compare

00000356 BRCS PC-0x2C Branch if carry set

00000357 RJMP PC+0x0005 Relative jump

295: g\_nSlots = (EEPROM\_SZ - STATS\_SZ) / NDE\_SZ;

00000358 LDI R24,0x3E Load immediate

00000359 STS 0x0232,R24 Store direct to data space

0000035B RJMP PC-0x0035 Relative jump

312: }

0000035C ADIW R28,0x22 Add immediate to word

0000035D IN R0,0x3F In from I/O location

0000035E CLI Global Interrupt Disable

0000035F OUT 0x3E,R29 Out to I/O location

00000360 OUT 0x3F,R0 Out to I/O location

00000361 OUT 0x3D,R28 Out to I/O location

00000362 POP R29 Pop register from stack

00000363 POP R28 Pop register from stack

00000364 POP R17 Pop register from stack

00000365 POP R16 Pop register from stack

00000366 POP R15 Pop register from stack

00000367 POP R14 Pop register from stack

00000368 POP R13 Pop register from stack

00000369 POP R12 Pop register from stack

0000036A POP R11 Pop register from stack

0000036B RET Subroutine return

316: {

0000036C PUSH R2 Push register on stack

0000036D PUSH R3 Push register on stack

0000036E PUSH R4 Push register on stack

0000036F PUSH R5 Push register on stack

00000370 PUSH R6 Push register on stack

00000371 PUSH R7 Push register on stack

00000372 PUSH R8 Push register on stack

00000373 PUSH R9 Push register on stack

00000374 PUSH R10 Push register on stack

00000375 PUSH R11 Push register on stack

00000376 PUSH R12 Push register on stack

00000377 PUSH R13 Push register on stack

00000378 PUSH R14 Push register on stack

00000379 PUSH R15 Push register on stack

0000037A PUSH R16 Push register on stack

0000037B PUSH R17 Push register on stack

0000037C PUSH R28 Push register on stack

0000037D PUSH R29 Push register on stack

0000037E IN R28,0x3D In from I/O location

0000037F IN R29,0x3E In from I/O location

00000380 SBIW R28,0x22 Subtract immediate from word

00000381 IN R0,0x3F In from I/O location

00000382 CLI Global Interrupt Disable

00000383 OUT 0x3E,R29 Out to I/O location

00000384 OUT 0x3F,R0 Out to I/O location

00000385 OUT 0x3D,R28 Out to I/O location

00000386 STD Y+17,R10 Store indirect with displacement

00000387 STD Y+18,R11 Store indirect with displacement

00000388 STD Y+19,R12 Store indirect with displacement

00000389 STD Y+20,R13 Store indirect with displacement

0000038A STD Y+21,R14 Store indirect with displacement

0000038B STD Y+22,R15 Store indirect with displacement

0000038C STD Y+23,R16 Store indirect with displacement

0000038D STD Y+24,R17 Store indirect with displacement

0000038E STD Y+25,R18 Store indirect with displacement

0000038F STD Y+26,R19 Store indirect with displacement

00000390 STD Y+27,R20 Store indirect with displacement

00000391 STD Y+28,R21 Store indirect with displacement

00000392 STD Y+29,R22 Store indirect with displacement

00000393 STD Y+30,R23 Store indirect with displacement

00000394 STD Y+31,R24 Store indirect with displacement

00000395 STD Y+32,R25 Store indirect with displacement

319: for (i = 0; i < g\_nSlots; i++)

00000396 LDS R24,0x0232 Load direct from data space

00000398 TST R24 Test for Zero or Minus

00000399 BRNE PC+0x02 Branch if not equal

0000039A RJMP PC+0x005B Relative jump

0000039B STD Y+34,R9 Store indirect with displacement

0000039C STD Y+33,R8 Store indirect with displacement

0000039D MOV R8,R1 Copy register

0000039E MOV R9,R1 Copy register

0000039F MOV R6,R1 Copy register

000003A0 MOV R7,R1 Copy register

321: uint32\_t addr = gEEPROM + i \* NDE\_SZ;

000003A1 LDS R24,0x0432 Load direct from data space

000003A3 LDS R25,0x0433 Load direct from data space

000003A5 LDS R26,0x0434 Load direct from data space

000003A7 LDS R27,0x0435 Load direct from data space

000003A9 MOVW R2,R24 Copy register pair

000003AA MOVW R4,R26 Copy register pair

000003AB ADD R2,R8 Add without carry

000003AC ADC R3,R9 Add with carry

000003AD ADC R4,R1 Add with carry

000003AE ADC R5,R1 Add with carry

322: EEPROM\_read(addr, &temp, NDE\_SZ);

000003AF LDI R18,0x10 Load immediate

000003B0 MOVW R20,R28 Copy register pair

000003B1 SUBI R20,0xFF Subtract immediate

000003B2 SBCI R21,0xFF Subtract immediate with carry

000003B3 MOVW R24,R4 Copy register pair

000003B4 MOVW R22,R2 Copy register pair

000003B5 RCALL PC-0x0233 Relative call subroutine

324: if (func\_if(temp, x))

000003B6 IN R18,0x3D In from I/O location

000003B7 IN R19,0x3E In from I/O location

000003B8 SUBI R18,0x10 Subtract immediate

000003B9 SBC R19,R1 Subtract with carry

000003BA IN R0,0x3F In from I/O location

000003BB CLI Global Interrupt Disable

000003BC OUT 0x3E,R19 Out to I/O location

000003BD OUT 0x3F,R0 Out to I/O location

000003BE OUT 0x3D,R18 Out to I/O location

000003BF IN R26,0x3D In from I/O location

000003C0 IN R27,0x3E In from I/O location

000003C1 ADIW R26,0x01 Add immediate to word

000003C2 LDI R24,0x10 Load immediate

000003C3 MOVW R30,R28 Copy register pair

000003C4 ADIW R30,0x11 Add immediate to word

000003C5 LD R0,Z+ Load indirect and postincrement

000003C6 ST X+,R0 Store indirect and postincrement

000003C7 DEC R24 Decrement

000003C8 BRNE PC-0x03 Branch if not equal

000003C9 LDD R10,Y+1 Load indirect with displacement

000003CA LDD R11,Y+2 Load indirect with displacement

000003CB LDD R12,Y+3 Load indirect with displacement

000003CC LDD R13,Y+4 Load indirect with displacement

000003CD LDD R14,Y+5 Load indirect with displacement

000003CE LDD R15,Y+6 Load indirect with displacement

000003CF LDD R16,Y+7 Load indirect with displacement

000003D0 LDD R17,Y+8 Load indirect with displacement

000003D1 LDD R18,Y+9 Load indirect with displacement

000003D2 LDD R19,Y+10 Load indirect with displacement

000003D3 LDD R20,Y+11 Load indirect with displacement

000003D4 LDD R21,Y+12 Load indirect with displacement

000003D5 LDD R22,Y+13 Load indirect with displacement

000003D6 LDD R23,Y+14 Load indirect with displacement

000003D7 LDD R24,Y+15 Load indirect with displacement

000003D8 LDD R25,Y+16 Load indirect with displacement

000003D9 LDD R30,Y+33 Load indirect with displacement

000003DA LDD R31,Y+34 Load indirect with displacement

000003DB ICALL Indirect call to (Z)

000003DC IN R0,0x3F In from I/O location

000003DD CLI Global Interrupt Disable

000003DE OUT 0x3E,R29 Out to I/O location

000003DF OUT 0x3F,R0 Out to I/O location

000003E0 OUT 0x3D,R28 Out to I/O location

000003E1 CPSE R24,R1 Compare, skip if equal

000003E2 RJMP PC+0x0018 Relative jump

319: for (i = 0; i < g\_nSlots; i++)

000003E3 SER R19 Set Register

000003E4 SUB R6,R19 Subtract without carry

000003E5 SBC R7,R19 Subtract with carry

000003E6 LDI R24,0x10 Load immediate

000003E7 ADD R8,R24 Add without carry

000003E8 ADC R9,R1 Add with carry

000003E9 LDS R24,0x0232 Load direct from data space

000003EB LDI R25,0x00 Load immediate

000003EC CP R6,R24 Compare

000003ED CPC R7,R25 Compare with carry

000003EE BRGE PC+0x02 Branch if greater or equal, signed

000003EF RJMP PC-0x004E Relative jump

327: return (uint32\_t)-1;

000003F0 SER R22 Set Register

000003F1 SER R23 Set Register

000003F2 SER R24 Set Register

000003F3 SER R25 Set Register

000003F4 RJMP PC+0x000A Relative jump

000003F5 SER R22 Set Register

000003F6 SER R23 Set Register

000003F7 SER R24 Set Register

000003F8 SER R25 Set Register

000003F9 RJMP PC+0x0005 Relative jump

321: uint32\_t addr = gEEPROM + i \* NDE\_SZ;

000003FA MOV R22,R2 Copy register

000003FB MOV R23,R3 Copy register

000003FC MOV R24,R4 Copy register

000003FD MOV R25,R5 Copy register

328: }

000003FE ADIW R28,0x22 Add immediate to word

000003FF IN R0,0x3F In from I/O location

00000400 CLI Global Interrupt Disable

00000401 OUT 0x3E,R29 Out to I/O location

00000402 OUT 0x3F,R0 Out to I/O location

00000403 OUT 0x3D,R28 Out to I/O location

00000404 POP R29 Pop register from stack

00000405 POP R28 Pop register from stack

00000406 POP R17 Pop register from stack

00000407 POP R16 Pop register from stack

00000408 POP R15 Pop register from stack

00000409 POP R14 Pop register from stack

0000040A POP R13 Pop register from stack

0000040B POP R12 Pop register from stack

0000040C POP R11 Pop register from stack

0000040D POP R10 Pop register from stack

0000040E POP R9 Pop register from stack

0000040F POP R8 Pop register from stack

00000410 POP R7 Pop register from stack

00000411 POP R6 Pop register from stack

00000412 POP R5 Pop register from stack

00000413 POP R4 Pop register from stack

00000414 POP R3 Pop register from stack

00000415 POP R2 Pop register from stack

00000416 RET Subroutine return

434: {

00000417 PUSH R6 Push register on stack

00000418 PUSH R7 Push register on stack

00000419 PUSH R8 Push register on stack

0000041A PUSH R9 Push register on stack

0000041B PUSH R10 Push register on stack

0000041C PUSH R11 Push register on stack

0000041D PUSH R12 Push register on stack

0000041E PUSH R13 Push register on stack

0000041F PUSH R14 Push register on stack

00000420 PUSH R15 Push register on stack

00000421 PUSH R16 Push register on stack

00000422 PUSH R17 Push register on stack

00000423 PUSH R28 Push register on stack

00000424 PUSH R29 Push register on stack

00000425 IN R28,0x3D In from I/O location

00000426 IN R29,0x3E In from I/O location

00000427 SBIW R28,0x10 Subtract immediate from word

00000428 IN R0,0x3F In from I/O location

00000429 CLI Global Interrupt Disable

0000042A OUT 0x3E,R29 Out to I/O location

0000042B OUT 0x3F,R0 Out to I/O location

0000042C OUT 0x3D,R28 Out to I/O location

0000042D MOVW R6,R22 Copy register pair

435: NDE to\_add = \*(NDE\*)(str + 2);

0000042E LDI R18,0x10 Load immediate

0000042F MOVW R30,R24 Copy register pair

00000430 ADIW R30,0x02 Add immediate to word

00000431 MOVW R26,R28 Copy register pair

00000432 ADIW R26,0x01 Add immediate to word

00000433 LD R0,Z+ Load indirect and postincrement

00000434 ST X+,R0 Store indirect and postincrement

00000435 DEC R18 Decrement

00000436 BRNE PC-0x03 Branch if not equal

436: to\_add.magic = EEPROM\_magic;

00000437 LDI R24,0xFC Load immediate

00000438 MOV R10,R24 Copy register

00000439 STD Y+1,R24 Store indirect with displacement

0000043A LDI R24,0xFD Load immediate

0000043B STD Y+2,R24 Store indirect with displacement

0000043C LDI R24,0xFE Load immediate

0000043D STD Y+3,R24 Store indirect with displacement

0000043E LDI R24,0xFA Load immediate

0000043F STD Y+4,R24 Store indirect with displacement

438: uint32\_t empty\_slot\_addr = find\_if(to\_add, slot\_empty);

00000440 MOV R0,R31 Copy register

00000441 LDI R31,0x8C Load immediate

00000442 MOV R8,R31 Copy register

00000443 LDI R31,0x00 Load immediate

00000444 MOV R9,R31 Copy register

00000445 MOV R31,R0 Copy register

00000446 MOV R0,R31 Copy register

00000447 LDI R31,0xFD Load immediate

00000448 MOV R11,R31 Copy register

00000449 MOV R31,R0 Copy register

0000044A MOV R0,R31 Copy register

0000044B LDI R31,0xFE Load immediate

0000044C MOV R12,R31 Copy register

0000044D MOV R31,R0 Copy register

0000044E MOV R0,R31 Copy register

0000044F LDI R31,0xFA Load immediate

00000450 MOV R13,R31 Copy register

00000451 MOV R31,R0 Copy register

00000452 LDD R14,Y+5 Load indirect with displacement

00000453 LDD R15,Y+6 Load indirect with displacement

00000454 LDD R16,Y+7 Load indirect with displacement

00000455 LDD R17,Y+8 Load indirect with displacement

00000456 LDD R18,Y+9 Load indirect with displacement

00000457 LDD R19,Y+10 Load indirect with displacement

00000458 LDD R20,Y+11 Load indirect with displacement

00000459 LDD R21,Y+12 Load indirect with displacement

0000045A LDD R22,Y+13 Load indirect with displacement

0000045B LDD R23,Y+14 Load indirect with displacement

0000045C LDD R24,Y+15 Load indirect with displacement

0000045D LDD R25,Y+16 Load indirect with displacement

0000045E RCALL PC-0x00F2 Relative call subroutine

440: if (empty\_slot\_addr != -1)

0000045F CPI R22,0xFF Compare with immediate

00000460 SER R18 Set Register

00000461 CPC R23,R18 Compare with carry

00000462 CPC R24,R18 Compare with carry

00000463 CPC R25,R18 Compare with carry

00000464 BREQ PC+0x0F Branch if equal

442: EEPROM\_write(empty\_slot\_addr, &to\_add, NDE\_SZ);

00000465 LDI R18,0x10 Load immediate

00000466 MOVW R20,R28 Copy register pair

00000467 SUBI R20,0xFF Subtract immediate

00000468 SBCI R21,0xFF Subtract immediate with carry

00000469 RCALL PC-0x037D Relative call subroutine

443: strcpy(out\_str, "11Parameters are added to container");

0000046A LDI R24,0x24 Load immediate

0000046B LDI R30,0x02 Load immediate

0000046C LDI R31,0x01 Load immediate

0000046D MOVW R26,R6 Copy register pair

0000046E LD R0,Z+ Load indirect and postincrement

0000046F ST X+,R0 Store indirect and postincrement

00000470 DEC R24 Decrement

00000471 BRNE PC-0x03 Branch if not equal

00000472 RJMP PC+0x0009 Relative jump

447: strcpy(out\_str, "00No free space in EEPROM");

00000473 LDI R24,0x1A Load immediate

00000474 LDI R30,0x26 Load immediate

00000475 LDI R31,0x01 Load immediate

00000476 MOVW R26,R6 Copy register pair

00000477 LD R0,Z+ Load indirect and postincrement

00000478 ST X+,R0 Store indirect and postincrement

00000479 DEC R24 Decrement

0000047A BRNE PC-0x03 Branch if not equal

448: }

0000047B ADIW R28,0x10 Add immediate to word

0000047C IN R0,0x3F In from I/O location

0000047D CLI Global Interrupt Disable

0000047E OUT 0x3E,R29 Out to I/O location

0000047F OUT 0x3F,R0 Out to I/O location

00000480 OUT 0x3D,R28 Out to I/O location

00000481 POP R29 Pop register from stack

00000482 POP R28 Pop register from stack

00000483 POP R17 Pop register from stack

00000484 POP R16 Pop register from stack

00000485 POP R15 Pop register from stack

00000486 POP R14 Pop register from stack

00000487 POP R13 Pop register from stack

00000488 POP R12 Pop register from stack

00000489 POP R11 Pop register from stack

0000048A POP R10 Pop register from stack

0000048B POP R9 Pop register from stack

0000048C POP R8 Pop register from stack

0000048D POP R7 Pop register from stack

0000048E POP R6 Pop register from stack

0000048F RET Subroutine return

452: {

00000490 PUSH R6 Push register on stack

00000491 PUSH R7 Push register on stack

00000492 PUSH R8 Push register on stack

00000493 PUSH R9 Push register on stack

00000494 PUSH R10 Push register on stack

00000495 PUSH R11 Push register on stack

00000496 PUSH R12 Push register on stack

00000497 PUSH R13 Push register on stack

00000498 PUSH R14 Push register on stack

00000499 PUSH R15 Push register on stack

0000049A PUSH R16 Push register on stack

0000049B PUSH R17 Push register on stack

0000049C PUSH R28 Push register on stack

0000049D PUSH R29 Push register on stack

0000049E IN R28,0x3D In from I/O location

0000049F IN R29,0x3E In from I/O location

000004A0 SBIW R28,0x20 Subtract immediate from word

000004A1 IN R0,0x3F In from I/O location

000004A2 CLI Global Interrupt Disable

000004A3 OUT 0x3E,R29 Out to I/O location

000004A4 OUT 0x3F,R0 Out to I/O location

000004A5 OUT 0x3D,R28 Out to I/O location

000004A6 MOVW R6,R22 Copy register pair

453: NDE to\_del = \*(NDE\*)(str + 2);

000004A7 LDI R18,0x10 Load immediate

000004A8 MOVW R30,R24 Copy register pair

000004A9 ADIW R30,0x02 Add immediate to word

000004AA MOVW R26,R28 Copy register pair

000004AB ADIW R26,0x01 Add immediate to word

000004AC LD R0,Z+ Load indirect and postincrement

000004AD ST X+,R0 Store indirect and postincrement

000004AE DEC R18 Decrement

000004AF BRNE PC-0x03 Branch if not equal

454: uint32\_t del\_nde\_addr = find\_if(to\_del, e\_equal);

000004B0 MOV R0,R31 Copy register

000004B1 LDI R31,0x47 Load immediate

000004B2 MOV R8,R31 Copy register

000004B3 LDI R31,0x00 Load immediate

000004B4 MOV R9,R31 Copy register

000004B5 MOV R31,R0 Copy register

000004B6 LDD R10,Y+1 Load indirect with displacement

000004B7 LDD R11,Y+2 Load indirect with displacement

000004B8 LDD R12,Y+3 Load indirect with displacement

000004B9 LDD R13,Y+4 Load indirect with displacement

000004BA LDD R14,Y+5 Load indirect with displacement

000004BB LDD R15,Y+6 Load indirect with displacement

000004BC LDD R16,Y+7 Load indirect with displacement

000004BD LDD R17,Y+8 Load indirect with displacement

000004BE LDD R18,Y+9 Load indirect with displacement

000004BF LDD R19,Y+10 Load indirect with displacement

000004C0 LDD R20,Y+11 Load indirect with displacement

000004C1 LDD R21,Y+12 Load indirect with displacement

000004C2 LDD R22,Y+13 Load indirect with displacement

000004C3 LDD R23,Y+14 Load indirect with displacement

000004C4 LDD R24,Y+15 Load indirect with displacement

000004C5 LDD R25,Y+16 Load indirect with displacement

000004C6 RCALL PC-0x015A Relative call subroutine

456: if (del\_nde\_addr != (uint32\_t)-1)

000004C7 CPI R22,0xFF Compare with immediate

000004C8 SER R18 Set Register

000004C9 CPC R23,R18 Compare with carry

000004CA CPC R24,R18 Compare with carry

000004CB CPC R25,R18 Compare with carry

000004CC BREQ PC+0x14 Branch if equal

458: NDE empty = { 0 };

000004CD MOVW R20,R28 Copy register pair

000004CE SUBI R20,0xEF Subtract immediate

000004CF SBCI R21,0xFF Subtract immediate with carry

000004D0 LDI R18,0x10 Load immediate

000004D1 MOVW R30,R20 Copy register pair

000004D2 ST Z+,R1 Store indirect and postincrement

000004D3 DEC R18 Decrement

459: EEPROM\_write(del\_nde\_addr, &empty, NDE\_SZ);

000004D4 BRNE PC-0x02 Branch if not equal

000004D5 LDI R18,0x10 Load immediate

460: strcpy(out\_str, "11Parameters are deleted from container");

000004D6 RCALL PC-0x03EA Relative call subroutine

000004D7 LDI R24,0x28 Load immediate

000004D8 LDI R30,0x40 Load immediate

000004D9 LDI R31,0x01 Load immediate

000004DA MOVW R26,R6 Copy register pair

000004DB LD R0,Z+ Load indirect and postincrement

000004DC ST X+,R0 Store indirect and postincrement

000004DD DEC R24 Decrement

000004DE BRNE PC-0x03 Branch if not equal

464: strcpy(out\_str, "00Parameters are not found in container");

000004DF RJMP PC+0x0009 Relative jump

000004E0 LDI R24,0x28 Load immediate

000004E1 LDI R30,0x68 Load immediate

000004E2 LDI R31,0x01 Load immediate

000004E3 MOVW R26,R6 Copy register pair

000004E4 LD R0,Z+ Load indirect and postincrement

000004E5 ST X+,R0 Store indirect and postincrement

000004E6 DEC R24 Decrement

466: }

000004E7 BRNE PC-0x03 Branch if not equal

000004E8 ADIW R28,0x20 Add immediate to word

000004E9 IN R0,0x3F In from I/O location

000004EA CLI Global Interrupt Disable

000004EB OUT 0x3E,R29 Out to I/O location

000004EC OUT 0x3F,R0 Out to I/O location

000004ED OUT 0x3D,R28 Out to I/O location

000004EE POP R29 Pop register from stack

000004EF POP R28 Pop register from stack

000004F0 POP R17 Pop register from stack

000004F1 POP R16 Pop register from stack

000004F2 POP R15 Pop register from stack

000004F3 POP R14 Pop register from stack

000004F4 POP R13 Pop register from stack

000004F5 POP R12 Pop register from stack

000004F6 POP R11 Pop register from stack

000004F7 POP R10 Pop register from stack

000004F8 POP R9 Pop register from stack

000004F9 POP R8 Pop register from stack

000004FA POP R7 Pop register from stack

000004FB POP R6 Pop register from stack

--- Нет исходного файла --------------------------------------------------------

000004FC RET Subroutine return

000004FD PUSH R6 Push register on stack

000004FE PUSH R7 Push register on stack

000004FF PUSH R8 Push register on stack

00000500 PUSH R9 Push register on stack

00000501 PUSH R10 Push register on stack

00000502 PUSH R11 Push register on stack

00000503 PUSH R12 Push register on stack

00000504 PUSH R13 Push register on stack

00000505 PUSH R14 Push register on stack

00000506 PUSH R15 Push register on stack

00000507 PUSH R16 Push register on stack

00000508 PUSH R17 Push register on stack

00000509 PUSH R28 Push register on stack

0000050A PUSH R29 Push register on stack

0000050B IN R28,0x3D In from I/O location

0000050C IN R29,0x3E In from I/O location

0000050D SBIW R28,0x10 Subtract immediate from word

0000050E IN R0,0x3F In from I/O location

0000050F CLI Global Interrupt Disable

00000510 OUT 0x3E,R29 Out to I/O location

00000511 OUT 0x3F,R0 Out to I/O location

00000512 OUT 0x3D,R28 Out to I/O location

00000513 MOVW R6,R22 Copy register pair

488: NDE x = \*(NDE\*)(str + 2);

00000514 LDI R18,0x10 Load immediate

00000515 MOVW R30,R24 Copy register pair

00000516 ADIW R30,0x02 Add immediate to word

00000517 MOVW R26,R28 Copy register pair

00000518 ADIW R26,0x01 Add immediate to word

00000519 LD R0,Z+ Load indirect and postincrement

0000051A ST X+,R0 Store indirect and postincrement

0000051B DEC R18 Decrement

0000051C BRNE PC-0x03 Branch if not equal

489: uint32\_t secret\_addr = find\_if(x, e\_equal);

0000051D MOV R0,R31 Copy register

0000051E LDI R31,0x47 Load immediate

0000051F MOV R8,R31 Copy register

00000520 LDI R31,0x00 Load immediate

00000521 MOV R9,R31 Copy register

00000522 MOV R31,R0 Copy register

00000523 LDD R10,Y+1 Load indirect with displacement

00000524 LDD R11,Y+2 Load indirect with displacement

00000525 LDD R12,Y+3 Load indirect with displacement

00000526 LDD R13,Y+4 Load indirect with displacement

00000527 LDD R14,Y+5 Load indirect with displacement

00000528 LDD R15,Y+6 Load indirect with displacement

00000529 LDD R16,Y+7 Load indirect with displacement

0000052A LDD R17,Y+8 Load indirect with displacement

0000052B LDD R18,Y+9 Load indirect with displacement

0000052C LDD R19,Y+10 Load indirect with displacement

0000052D LDD R20,Y+11 Load indirect with displacement

0000052E LDD R21,Y+12 Load indirect with displacement

0000052F LDD R22,Y+13 Load indirect with displacement

00000530 LDD R23,Y+14 Load indirect with displacement

00000531 LDD R24,Y+15 Load indirect with displacement

00000532 LDD R25,Y+16 Load indirect with displacement

00000533 RCALL PC-0x01C7 Relative call subroutine

492: if (secret\_addr != -1)

00000534 CPI R22,0xFF Compare with immediate

00000535 SER R18 Set Register

00000536 CPC R23,R18 Compare with carry

00000537 CPC R24,R18 Compare with carry

00000538 CPC R25,R18 Compare with carry

00000539 BREQ PC+0x0E Branch if equal

494: EEPROM\_read(secret\_addr, &gCurNde, NDE\_SZ);

0000053A LDI R18,0x10 Load immediate

0000053B LDI R20,0x22 Load immediate

0000053C LDI R21,0x02 Load immediate

0000053D RCALL PC-0x03BB Relative call subroutine

495: strcpy(out\_str, "11Encryption parameters are set");

0000053E LDI R24,0x20 Load immediate

0000053F LDI R30,0x90 Load immediate

00000540 LDI R31,0x01 Load immediate

00000541 MOVW R26,R6 Copy register pair

00000542 LD R0,Z+ Load indirect and postincrement

00000543 ST X+,R0 Store indirect and postincrement

00000544 DEC R24 Decrement

00000545 BRNE PC-0x03 Branch if not equal

00000546 RJMP PC+0x0009 Relative jump

498: strcpy(out\_str, "00Encryption parameters weren't found");

00000547 LDI R24,0x26 Load immediate

00000548 LDI R30,0xB0 Load immediate

00000549 LDI R31,0x01 Load immediate

0000054A MOVW R26,R6 Copy register pair

0000054B LD R0,Z+ Load indirect and postincrement

0000054C ST X+,R0 Store indirect and postincrement

0000054D DEC R24 Decrement

0000054E BRNE PC-0x03 Branch if not equal

499: }

0000054F ADIW R28,0x10 Add immediate to word

00000550 IN R0,0x3F In from I/O location

00000551 CLI Global Interrupt Disable

00000552 OUT 0x3E,R29 Out to I/O location

00000553 OUT 0x3F,R0 Out to I/O location

00000554 OUT 0x3D,R28 Out to I/O location

00000555 POP R29 Pop register from stack

00000556 POP R28 Pop register from stack

00000557 POP R17 Pop register from stack

00000558 POP R16 Pop register from stack

00000559 POP R15 Pop register from stack

0000055A POP R14 Pop register from stack

0000055B POP R13 Pop register from stack

0000055C POP R12 Pop register from stack

0000055D POP R11 Pop register from stack

0000055E POP R10 Pop register from stack

0000055F POP R9 Pop register from stack

00000560 POP R8 Pop register from stack

00000561 POP R7 Pop register from stack

00000562 POP R6 Pop register from stack

00000563 RET Subroutine return

341: {

00000564 PUSH R4 Push register on stack

00000565 PUSH R5 Push register on stack

00000566 PUSH R6 Push register on stack

00000567 PUSH R7 Push register on stack

00000568 PUSH R8 Push register on stack

00000569 PUSH R9 Push register on stack

0000056A PUSH R10 Push register on stack

0000056B PUSH R11 Push register on stack

0000056C PUSH R14 Push register on stack

0000056D PUSH R15 Push register on stack

0000056E PUSH R16 Push register on stack

0000056F PUSH R17 Push register on stack

00000570 PUSH R28 Push register on stack

00000571 PUSH R29 Push register on stack

00000572 RCALL PC+0x0001 Relative call subroutine

00000573 RCALL PC+0x0001 Relative call subroutine

00000574 IN R28,0x3D In from I/O location

00000575 IN R29,0x3E In from I/O location

00000576 MOVW R4,R22 Copy register pair

00000577 MOVW R6,R24 Copy register pair

00000578 MOVW R8,R18 Copy register pair

00000579 MOVW R10,R20 Copy register pair

344: while (exp != 0)

0000057A OR R18,R19 Logical OR

0000057B OR R18,R20 Logical OR

0000057C OR R18,R21 Logical OR

0000057D BREQ PC+0x2D Branch if equal

0000057E LDI R24,0x01 Load immediate

0000057F LDI R25,0x00 Load immediate

00000580 LDI R26,0x00 Load immediate

00000581 LDI R27,0x00 Load immediate

00000582 STD Y+1,R24 Store indirect with displacement

00000583 STD Y+2,R25 Store indirect with displacement

00000584 STD Y+3,R26 Store indirect with displacement

00000585 STD Y+4,R27 Store indirect with displacement

346: if ((exp & 1) != 0)

00000586 SBRS R8,0 Skip if bit in register set

00000587 RJMP PC+0x000F Relative jump

348: res = (1u \* res \* base) % mod;

00000588 MOVW R20,R6 Copy register pair

00000589 MOVW R18,R4 Copy register pair

0000058A LDD R22,Y+1 Load indirect with displacement

0000058B LDD R23,Y+2 Load indirect with displacement

0000058C LDD R24,Y+3 Load indirect with displacement

0000058D LDD R25,Y+4 Load indirect with displacement

0000058E RCALL PC+0x0289 Relative call subroutine

0000058F MOVW R20,R16 Copy register pair

00000590 MOVW R18,R14 Copy register pair

00000591 RCALL PC+0x0295 Relative call subroutine

00000592 STD Y+1,R22 Store indirect with displacement

00000593 STD Y+2,R23 Store indirect with displacement

00000594 STD Y+3,R24 Store indirect with displacement

00000595 STD Y+4,R25 Store indirect with displacement

351: base = (1u \* base \* base) % mod;

00000596 MOVW R20,R6 Copy register pair

00000597 MOVW R18,R4 Copy register pair

00000598 MOVW R24,R6 Copy register pair

00000599 MOVW R22,R4 Copy register pair

0000059A RCALL PC+0x027D Relative call subroutine

0000059B MOVW R20,R16 Copy register pair

0000059C MOVW R18,R14 Copy register pair

0000059D RCALL PC+0x0289 Relative call subroutine

0000059E MOVW R4,R22 Copy register pair

0000059F MOVW R6,R24 Copy register pair

352: exp >>= 1;

000005A0 LSR R11 Logical shift right

000005A1 ROR R10 Rotate right through carry

000005A2 ROR R9 Rotate right through carry

000005A3 ROR R8 Rotate right through carry

344: while (exp != 0)

000005A4 CP R8,R1 Compare

000005A5 CPC R9,R1 Compare with carry

000005A6 CPC R10,R1 Compare with carry

000005A7 CPC R11,R1 Compare with carry

000005A8 BRNE PC-0x22 Branch if not equal

000005A9 RJMP PC+0x0009 Relative jump

342: uint32\_t res = 1;

000005AA LDI R24,0x01 Load immediate

000005AB LDI R25,0x00 Load immediate

000005AC LDI R26,0x00 Load immediate

000005AD LDI R27,0x00 Load immediate

000005AE STD Y+1,R24 Store indirect with displacement

000005AF STD Y+2,R25 Store indirect with displacement

000005B0 STD Y+3,R26 Store indirect with displacement

000005B1 STD Y+4,R27 Store indirect with displacement

356: }

000005B2 LDD R22,Y+1 Load indirect with displacement

000005B3 LDD R23,Y+2 Load indirect with displacement

000005B4 LDD R24,Y+3 Load indirect with displacement

000005B5 LDD R25,Y+4 Load indirect with displacement

000005B6 POP R0 Pop register from stack

000005B7 POP R0 Pop register from stack

000005B8 POP R0 Pop register from stack

000005B9 POP R0 Pop register from stack

000005BA POP R29 Pop register from stack

000005BB POP R28 Pop register from stack

000005BC POP R17 Pop register from stack

000005BD POP R16 Pop register from stack

000005BE POP R15 Pop register from stack

000005BF POP R14 Pop register from stack

000005C0 POP R11 Pop register from stack

000005C1 POP R10 Pop register from stack

000005C2 POP R9 Pop register from stack

000005C3 POP R8 Pop register from stack

000005C4 POP R7 Pop register from stack

000005C5 POP R6 Pop register from stack

000005C6 POP R5 Pop register from stack

000005C7 POP R4 Pop register from stack

000005C8 RET Subroutine return

388: {

000005C9 PUSH R2 Push register on stack

000005CA PUSH R3 Push register on stack

000005CB PUSH R4 Push register on stack

000005CC PUSH R5 Push register on stack

000005CD PUSH R6 Push register on stack

000005CE PUSH R7 Push register on stack

000005CF PUSH R8 Push register on stack

000005D0 PUSH R9 Push register on stack

000005D1 PUSH R11 Push register on stack

000005D2 PUSH R12 Push register on stack

000005D3 PUSH R13 Push register on stack

000005D4 PUSH R14 Push register on stack

000005D5 PUSH R15 Push register on stack

000005D6 PUSH R16 Push register on stack

000005D7 PUSH R17 Push register on stack

000005D8 PUSH R28 Push register on stack

000005D9 PUSH R29 Push register on stack

000005DA IN R28,0x3D In from I/O location

000005DB IN R29,0x3E In from I/O location

000005DC SBIW R28,0x12 Subtract immediate from word

000005DD IN R0,0x3F In from I/O location

000005DE CLI Global Interrupt Disable

000005DF OUT 0x3E,R29 Out to I/O location

000005E0 OUT 0x3F,R0 Out to I/O location

000005E1 OUT 0x3D,R28 Out to I/O location

000005E2 MOVW R8,R24 Copy register pair

000005E3 MOVW R6,R22 Copy register pair

391: memcpy(out\_str, str, offs);

000005E4 LDI R24,0x06 Load immediate

000005E5 MOVW R30,R8 Copy register pair

000005E6 MOVW R26,R22 Copy register pair

000005E7 LD R0,Z+ Load indirect and postincrement

000005E8 ST X+,R0 Store indirect and postincrement

000005E9 DEC R24 Decrement

000005EA BRNE PC-0x03 Branch if not equal

392: uint32\_t nIters = \*(uint32\_t\*)(str + 2);

000005EB MOVW R30,R8 Copy register pair

000005EC LDD R2,Z+2 Load indirect with displacement

000005ED LDD R3,Z+3 Load indirect with displacement

000005EE LDD R4,Z+4 Load indirect with displacement

000005EF LDD R5,Z+5 Load indirect with displacement

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

399: for (i = 0; i < nIters; i++)

000005F0 CP R2,R1 Compare

000005F1 CPC R3,R1 Compare with carry

000005F2 CPC R4,R1 Compare with carry

000005F3 CPC R5,R1 Compare with carry

000005F4 BREQ PC+0x31 Branch if equal

000005F5 MOV R11,R1 Copy register

400: out\_buf[i] = powmod(buf[i], gCurNde.e, gCurNde.n);

000005F6 MOV R24,R11 Copy register

000005F7 LDI R25,0x00 Load immediate

000005F8 MOVW R12,R24 Copy register pair

000005F9 LSL R12 Logical Shift Left

000005FA ROL R13 Rotate Left Through Carry

000005FB LDI R31,0x06 Load immediate

000005FC ADD R12,R31 Add without carry

000005FD ADC R13,R1 Add with carry

000005FE ADD R12,R6 Add without carry

000005FF ADC R13,R7 Add with carry

00000600 LDS R14,0x0226 Load direct from data space

00000602 LDS R15,0x0227 Load direct from data space

00000604 LDS R16,0x0228 Load direct from data space

00000606 LDS R17,0x0229 Load direct from data space

00000608 LDS R18,0x022A Load direct from data space

0000060A LDS R19,0x022B Load direct from data space

0000060C LDS R20,0x022C Load direct from data space

0000060E LDS R21,0x022D Load direct from data space

00000610 MOVW R30,R8 Copy register pair

00000611 ADD R30,R24 Add without carry

00000612 ADC R31,R25 Add with carry

00000613 LDD R22,Z+6 Load indirect with displacement

00000614 LDI R23,0x00 Load immediate

00000615 LDI R24,0x00 Load immediate

00000616 LDI R25,0x00 Load immediate

00000617 RCALL PC-0x00B3 Relative call subroutine

00000618 MOVW R30,R12 Copy register pair

00000619 STD Z+1,R23 Store indirect with displacement

0000061A STD Z+0,R22 Store indirect with displacement

399: for (i = 0; i < nIters; i++)

0000061B INC R11 Increment

0000061C MOV R24,R11 Copy register

0000061D LDI R25,0x00 Load immediate

0000061E LDI R26,0x00 Load immediate

0000061F LDI R27,0x00 Load immediate

00000620 CP R24,R2 Compare

00000621 CPC R25,R3 Compare with carry

00000622 CPC R26,R4 Compare with carry

00000623 CPC R27,R5 Compare with carry

00000624 BRCS PC-0x2E Branch if carry set

404: EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

00000625 LDS R22,0x0436 Load direct from data space

00000627 LDS R23,0x0437 Load direct from data space

00000629 LDS R24,0x0438 Load direct from data space

0000062B LDS R25,0x0439 Load direct from data space

0000062D LDI R18,0x12 Load immediate

0000062E MOVW R20,R28 Copy register pair

0000062F SUBI R20,0xFF Subtract immediate

00000630 SBCI R21,0xFF Subtract immediate with carry

00000631 RCALL PC-0x04AF Relative call subroutine

405: st.nEncryptedBytes += nIters;

00000632 LDD R24,Y+11 Load indirect with displacement

00000633 LDD R25,Y+12 Load indirect with displacement

00000634 LDD R26,Y+13 Load indirect with displacement

00000635 LDD R27,Y+14 Load indirect with displacement

00000636 ADD R24,R2 Add without carry

00000637 ADC R25,R3 Add with carry

00000638 ADC R26,R4 Add with carry

00000639 ADC R27,R5 Add with carry

0000063A STD Y+11,R24 Store indirect with displacement

0000063B STD Y+12,R25 Store indirect with displacement

0000063C STD Y+13,R26 Store indirect with displacement

0000063D STD Y+14,R27 Store indirect with displacement

406: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

0000063E LDS R22,0x0436 Load direct from data space

00000640 LDS R23,0x0437 Load direct from data space

00000642 LDS R24,0x0438 Load direct from data space

00000644 LDS R25,0x0439 Load direct from data space

00000646 LDI R18,0x12 Load immediate

00000647 MOVW R20,R28 Copy register pair

00000648 SUBI R20,0xFF Subtract immediate

00000649 SBCI R21,0xFF Subtract immediate with carry

0000064A RCALL PC-0x055E Relative call subroutine

407: }

0000064B ADIW R28,0x12 Add immediate to word

0000064C IN R0,0x3F In from I/O location

0000064D CLI Global Interrupt Disable

0000064E OUT 0x3E,R29 Out to I/O location

0000064F OUT 0x3F,R0 Out to I/O location

00000650 OUT 0x3D,R28 Out to I/O location

00000651 POP R29 Pop register from stack

00000652 POP R28 Pop register from stack

00000653 POP R17 Pop register from stack

00000654 POP R16 Pop register from stack

00000655 POP R15 Pop register from stack

00000656 POP R14 Pop register from stack

00000657 POP R13 Pop register from stack

00000658 POP R12 Pop register from stack

00000659 POP R11 Pop register from stack

0000065A POP R9 Pop register from stack

0000065B POP R8 Pop register from stack

0000065C POP R7 Pop register from stack

0000065D POP R6 Pop register from stack

0000065E POP R5 Pop register from stack

0000065F POP R4 Pop register from stack

00000660 POP R3 Pop register from stack

00000661 POP R2 Pop register from stack

00000662 RET Subroutine return

411: {

00000663 PUSH R2 Push register on stack

00000664 PUSH R3 Push register on stack

00000665 PUSH R4 Push register on stack

00000666 PUSH R5 Push register on stack

00000667 PUSH R6 Push register on stack

00000668 PUSH R7 Push register on stack

00000669 PUSH R8 Push register on stack

0000066A PUSH R9 Push register on stack

0000066B PUSH R10 Push register on stack

0000066C PUSH R11 Push register on stack

0000066D PUSH R13 Push register on stack

0000066E PUSH R14 Push register on stack

0000066F PUSH R15 Push register on stack

00000670 PUSH R16 Push register on stack

00000671 PUSH R17 Push register on stack

00000672 PUSH R28 Push register on stack

00000673 PUSH R29 Push register on stack

00000674 IN R28,0x3D In from I/O location

00000675 IN R29,0x3E In from I/O location

00000676 SBIW R28,0x12 Subtract immediate from word

00000677 IN R0,0x3F In from I/O location

00000678 CLI Global Interrupt Disable

00000679 OUT 0x3E,R29 Out to I/O location

0000067A OUT 0x3F,R0 Out to I/O location

0000067B OUT 0x3D,R28 Out to I/O location

0000067C MOVW R8,R24 Copy register pair

0000067D MOVW R6,R22 Copy register pair

414: memcpy(out\_str, str, offs);

0000067E LDI R24,0x06 Load immediate

0000067F MOVW R30,R8 Copy register pair

00000680 MOVW R26,R22 Copy register pair

00000681 LD R0,Z+ Load indirect and postincrement

00000682 ST X+,R0 Store indirect and postincrement

00000683 DEC R24 Decrement

00000684 BRNE PC-0x03 Branch if not equal

415: uint32\_t nIters = \*(uint32\_t\*)(str + 2);

00000685 MOVW R30,R8 Copy register pair

00000686 LDD R2,Z+2 Load indirect with displacement

00000687 LDD R3,Z+3 Load indirect with displacement

00000688 LDD R4,Z+4 Load indirect with displacement

00000689 LDD R5,Z+5 Load indirect with displacement

422: for (i = 0; i < nIters; i++)

0000068A CP R2,R1 Compare

0000068B CPC R3,R1 Compare with carry

0000068C CPC R4,R1 Compare with carry

0000068D CPC R5,R1 Compare with carry

0000068E BREQ PC+0x30 Branch if equal

0000068F MOV R13,R1 Copy register

423: out\_buf[i] = powmod(buf[i], gCurNde.d, gCurNde.n);

00000690 MOV R24,R13 Copy register

00000691 LDI R25,0x00 Load immediate

00000692 MOVW R10,R24 Copy register pair

00000693 LDI R31,0x06 Load immediate

00000694 ADD R10,R31 Add without carry

00000695 ADC R11,R1 Add with carry

00000696 ADD R10,R6 Add without carry

00000697 ADC R11,R7 Add with carry

00000698 LDS R14,0x0226 Load direct from data space

0000069A LDS R15,0x0227 Load direct from data space

0000069C LDS R16,0x0228 Load direct from data space

0000069E LDS R17,0x0229 Load direct from data space

000006A0 LDS R18,0x022E Load direct from data space

000006A2 LDS R19,0x022F Load direct from data space

000006A4 LDS R20,0x0230 Load direct from data space

000006A6 LDS R21,0x0231 Load direct from data space

000006A8 LSL R24 Logical Shift Left

000006A9 ROL R25 Rotate Left Through Carry

000006AA MOVW R30,R8 Copy register pair

000006AB ADD R30,R24 Add without carry

000006AC ADC R31,R25 Add with carry

000006AD LDD R22,Z+6 Load indirect with displacement

000006AE LDD R23,Z+7 Load indirect with displacement

000006AF LDI R24,0x00 Load immediate

000006B0 LDI R25,0x00 Load immediate

000006B1 RCALL PC-0x014D Relative call subroutine

000006B2 MOVW R30,R10 Copy register pair

000006B3 STD Z+0,R22 Store indirect with displacement

422: for (i = 0; i < nIters; i++)

000006B4 INC R13 Increment

000006B5 MOV R24,R13 Copy register

000006B6 LDI R25,0x00 Load immediate

000006B7 LDI R26,0x00 Load immediate

000006B8 LDI R27,0x00 Load immediate

000006B9 CP R24,R2 Compare

000006BA CPC R25,R3 Compare with carry

000006BB CPC R26,R4 Compare with carry

000006BC CPC R27,R5 Compare with carry

000006BD BRCS PC-0x2D Branch if carry set

427: EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

000006BE LDS R22,0x0436 Load direct from data space

000006C0 LDS R23,0x0437 Load direct from data space

000006C2 LDS R24,0x0438 Load direct from data space

000006C4 LDS R25,0x0439 Load direct from data space

000006C6 LDI R18,0x12 Load immediate

000006C7 MOVW R20,R28 Copy register pair

000006C8 SUBI R20,0xFF Subtract immediate

000006C9 SBCI R21,0xFF Subtract immediate with carry

000006CA RCALL PC-0x0548 Relative call subroutine

428: st.nDecryptedBytes += nIters;

000006CB LDD R24,Y+15 Load indirect with displacement

000006CC LDD R25,Y+16 Load indirect with displacement

000006CD LDD R26,Y+17 Load indirect with displacement

000006CE LDD R27,Y+18 Load indirect with displacement

000006CF ADD R24,R2 Add without carry

000006D0 ADC R25,R3 Add with carry

000006D1 ADC R26,R4 Add with carry

000006D2 ADC R27,R5 Add with carry

000006D3 STD Y+15,R24 Store indirect with displacement

000006D4 STD Y+16,R25 Store indirect with displacement

000006D5 STD Y+17,R26 Store indirect with displacement

000006D6 STD Y+18,R27 Store indirect with displacement

429: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

000006D7 LDS R22,0x0436 Load direct from data space

000006D9 LDS R23,0x0437 Load direct from data space

000006DB LDS R24,0x0438 Load direct from data space

000006DD LDS R25,0x0439 Load direct from data space

000006DF LDI R18,0x12 Load immediate

000006E0 MOVW R20,R28 Copy register pair

000006E1 SUBI R20,0xFF Subtract immediate

000006E2 SBCI R21,0xFF Subtract immediate with carry

000006E3 RCALL PC-0x05F7 Relative call subroutine

430: }

000006E4 ADIW R28,0x12 Add immediate to word

000006E5 IN R0,0x3F In from I/O location

000006E6 CLI Global Interrupt Disable

000006E7 OUT 0x3E,R29 Out to I/O location

000006E8 OUT 0x3F,R0 Out to I/O location

000006E9 OUT 0x3D,R28 Out to I/O location

000006EA POP R29 Pop register from stack

000006EB POP R28 Pop register from stack

000006EC POP R17 Pop register from stack

000006ED POP R16 Pop register from stack

000006EE POP R15 Pop register from stack

000006EF POP R14 Pop register from stack

000006F0 POP R13 Pop register from stack

000006F1 POP R11 Pop register from stack

000006F2 POP R10 Pop register from stack

000006F3 POP R9 Pop register from stack

000006F4 POP R8 Pop register from stack

000006F5 POP R7 Pop register from stack

000006F6 POP R6 Pop register from stack

000006F7 POP R5 Pop register from stack

000006F8 POP R4 Pop register from stack

000006F9 POP R3 Pop register from stack

000006FA POP R2 Pop register from stack

000006FB RET Subroutine return

469: {

000006FC PUSH R28 Push register on stack

000006FD PUSH R29 Push register on stack

000006FE IN R28,0x3D In from I/O location

000006FF IN R29,0x3E In from I/O location

00000700 SBIW R28,0x12 Subtract immediate from word

00000701 IN R0,0x3F In from I/O location

00000702 CLI Global Interrupt Disable

00000703 OUT 0x3E,R29 Out to I/O location

00000704 OUT 0x3F,R0 Out to I/O location

00000705 OUT 0x3D,R28 Out to I/O location

472: EEPROM\_read(gStatistics\_addr, &st, STATS\_SZ);

00000706 LDS R22,0x0436 Load direct from data space

00000708 LDS R23,0x0437 Load direct from data space

0000070A LDS R24,0x0438 Load direct from data space

0000070C LDS R25,0x0439 Load direct from data space

0000070E LDI R18,0x12 Load immediate

0000070F MOVW R20,R28 Copy register pair

00000710 SUBI R20,0xFF Subtract immediate

00000711 SBCI R21,0xFF Subtract immediate with carry

00000712 RCALL PC-0x0590 Relative call subroutine

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

473: st.nRequestNoOpenKey++;

00000713 LDD R24,Y+9 Load indirect with displacement

00000714 LDD R25,Y+10 Load indirect with displacement

00000715 ADIW R24,0x01 Add immediate to word

00000716 STD Y+10,R25 Store indirect with displacement

00000717 STD Y+9,R24 Store indirect with displacement

474: EEPROM\_write(gStatistics\_addr, &st, STATS\_SZ);

00000718 LDS R22,0x0436 Load direct from data space

0000071A LDS R23,0x0437 Load direct from data space

0000071C LDS R24,0x0438 Load direct from data space

0000071E LDS R25,0x0439 Load direct from data space

00000720 LDI R18,0x12 Load immediate

00000721 MOVW R20,R28 Copy register pair

00000722 SUBI R20,0xFF Subtract immediate

00000723 SBCI R21,0xFF Subtract immediate with carry

00000724 RCALL PC-0x0638 Relative call subroutine

475: }

00000725 ADIW R28,0x12 Add immediate to word

00000726 IN R0,0x3F In from I/O location

00000727 CLI Global Interrupt Disable

00000728 OUT 0x3E,R29 Out to I/O location

00000729 OUT 0x3F,R0 Out to I/O location

0000072A OUT 0x3D,R28 Out to I/O location

0000072B POP R29 Pop register from stack

0000072C POP R28 Pop register from stack

0000072D RET Subroutine return

480: out\_str[0] = out\_str[1] = '1';

0000072E LDI R24,0x31 Load immediate

0000072F MOVW R30,R22 Copy register pair

00000730 STD Z+1,R24 Store indirect with displacement

00000731 STD Z+0,R24 Store indirect with displacement

481: EEPROM\_read(gStatistics\_addr, out\_str + 2, STATS\_SZ);

00000732 MOVW R20,R22 Copy register pair

00000733 SUBI R20,0xFE Subtract immediate

00000734 SBCI R21,0xFF Subtract immediate with carry

00000735 LDS R22,0x0436 Load direct from data space

00000737 LDS R23,0x0437 Load direct from data space

00000739 LDS R24,0x0438 Load direct from data space

0000073B LDS R25,0x0439 Load direct from data space

0000073D LDI R18,0x12 Load immediate

482: inc\_stat\_nook();

0000073E RCALL PC-0x05BC Relative call subroutine

0000073F RJMP PC-0x0043 Relative jump

00000740 RET Subroutine return

503: {

00000741 PUSH R14 Push register on stack

00000742 PUSH R15 Push register on stack

00000743 PUSH R16 Push register on stack

00000744 PUSH R17 Push register on stack

00000745 PUSH R28 Push register on stack

00000746 PUSH R29 Push register on stack

00000747 IN R28,0x3D In from I/O location

00000748 IN R29,0x3E In from I/O location

00000749 SBIW R28,0x10 Subtract immediate from word

0000074A IN R0,0x3F In from I/O location

0000074B CLI Global Interrupt Disable

0000074C OUT 0x3E,R29 Out to I/O location

0000074D OUT 0x3F,R0 Out to I/O location

0000074E OUT 0x3D,R28 Out to I/O location

0000074F MOVW R14,R22 Copy register pair

509: for (i = 0; i < g\_nSlots; i++)

00000750 LDS R24,0x0232 Load direct from data space

00000752 TST R24 Test for Zero or Minus

00000753 BREQ PC+0x37 Branch if equal

00000754 LDI R16,0x00 Load immediate

00000755 LDI R17,0x00 Load immediate

512: EEPROM\_read(addr, &Nde, NDE\_SZ);

00000756 LDI R24,0x10 Load immediate

00000757 MUL R17,R24 Multiply unsigned

00000758 MOVW R18,R0 Copy register pair

00000759 CLR R1 Clear Register

0000075A LDS R24,0x0432 Load direct from data space

0000075C LDS R25,0x0433 Load direct from data space

0000075E LDS R26,0x0434 Load direct from data space

00000760 LDS R27,0x0435 Load direct from data space

00000762 MOVW R22,R24 Copy register pair

00000763 MOVW R24,R26 Copy register pair

00000764 ADD R22,R18 Add without carry

00000765 ADC R23,R19 Add with carry

00000766 ADC R24,R1 Add with carry

00000767 ADC R25,R1 Add with carry

00000768 LDI R18,0x10 Load immediate

00000769 MOVW R20,R28 Copy register pair

0000076A SUBI R20,0xFF Subtract immediate

0000076B SBCI R21,0xFF Subtract immediate with carry

0000076C RCALL PC-0x05EA Relative call subroutine

514: if (Nde.magic == EEPROM\_magic)

0000076D LDD R24,Y+1 Load indirect with displacement

0000076E LDD R25,Y+2 Load indirect with displacement

0000076F LDD R26,Y+3 Load indirect with displacement

00000770 LDD R27,Y+4 Load indirect with displacement

00000771 CPI R24,0xFC Compare with immediate

00000772 SBCI R25,0xFD Subtract immediate with carry

00000773 SBCI R26,0xFE Subtract immediate with carry

00000774 SBCI R27,0xFA Subtract immediate with carry

00000775 BRNE PC+0x10 Branch if not equal

515: out[k++] = Nde.e;

00000776 MOVW R30,R14 Copy register pair

00000777 LDI R24,0x04 Load immediate

00000778 MUL R16,R24 Multiply unsigned

00000779 ADD R30,R0 Add without carry

0000077A ADC R31,R1 Add with carry

0000077B CLR R1 Clear Register

0000077C LDD R24,Y+9 Load indirect with displacement

0000077D LDD R25,Y+10 Load indirect with displacement

0000077E LDD R26,Y+11 Load indirect with displacement

0000077F LDD R27,Y+12 Load indirect with displacement

00000780 STD Z+0,R24 Store indirect with displacement

00000781 STD Z+1,R25 Store indirect with displacement

00000782 STD Z+2,R26 Store indirect with displacement

00000783 STD Z+3,R27 Store indirect with displacement

00000784 SUBI R16,0xFF Subtract immediate

509: for (i = 0; i < g\_nSlots; i++)

00000785 SUBI R17,0xFF Subtract immediate

00000786 LDS R24,0x0232 Load direct from data space

00000788 CP R17,R24 Compare

517: inc\_stat\_nook();

00000789 BRCS PC-0x33 Branch if carry set

0000078A RCALL PC-0x008E Relative call subroutine

518: }

0000078B ADIW R28,0x10 Add immediate to word

0000078C IN R0,0x3F In from I/O location

0000078D CLI Global Interrupt Disable

0000078E OUT 0x3E,R29 Out to I/O location

0000078F OUT 0x3F,R0 Out to I/O location

00000790 OUT 0x3D,R28 Out to I/O location

00000791 POP R29 Pop register from stack

00000792 POP R28 Pop register from stack

00000793 POP R17 Pop register from stack

00000794 POP R16 Pop register from stack

00000795 POP R15 Pop register from stack

00000796 POP R14 Pop register from stack

00000797 RET Subroutine return

539: {

00000798 PUSH R28 Push register on stack

00000799 PUSH R29 Push register on stack

0000079A IN R28,0x3D In from I/O location

0000079B IN R29,0x3E In from I/O location

0000079C SUBI R28,0xFC Subtract immediate

0000079D SBCI R29,0x01 Subtract immediate with carry

0000079E IN R0,0x3F In from I/O location

0000079F CLI Global Interrupt Disable

000007A0 OUT 0x3E,R29 Out to I/O location

000007A1 OUT 0x3F,R0 Out to I/O location

000007A2 OUT 0x3D,R28 Out to I/O location

540: DDRA = 0xFF;

000007A3 SER R17 Set Register

542: EEPROM\_init();

000007A4 OUT 0x1A,R17 Out to I/O location

543: PORTA = 0x01;

000007A5 RCALL PC-0x06E7 Relative call subroutine

545: CryptInit();

000007A6 LDI R24,0x01 Load immediate

546: PORTA = 0x02;

000007A7 OUT 0x1B,R24 Out to I/O location

548: USART\_Init();

000007A8 RCALL PC-0x04D5 Relative call subroutine

000007A9 LDI R24,0x02 Load immediate

549: PORTA = 0xFF;

000007AA OUT 0x1B,R24 Out to I/O location

--- c:\program files (x86)\atmel\studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\util/delay.h

187: \_\_builtin\_avr\_delay\_cycles(\_\_ticks\_dc);

000007AB RCALL PC-0x0590 Relative call subroutine

000007AC OUT 0x1B,R17 Out to I/O location

000007AD SER R18 Set Register

000007AE LDI R24,0x34 Load immediate

000007AF LDI R25,0x0C Load immediate

000007B0 SUBI R18,0x01 Subtract immediate

000007B1 SBCI R24,0x00 Subtract immediate with carry

--- c:\program files (x86)\atmel\studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\util/delay.h

000007B2 SBCI R25,0x00 Subtract immediate with carry

000007B3 BRNE PC-0x03 Branch if not equal

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

552: PORTA = 0;

000007B4 RJMP PC+0x0001 Relative jump

554: sei();

000007B5 NOP No operation

561: char readBuf[BUF\_SZ] = { 0 }, writeBuf[BUF\_SZ] = { 0 };

000007B6 OUT 0x1B,R1 Out to I/O location

000007B7 SEI Global Interrupt Enable

000007B8 MOVW R16,R28 Copy register pair

000007B9 SUBI R16,0x01 Subtract immediate

000007BA SBCI R17,0xFF Subtract immediate with carry

000007BB MOV R0,R31 Copy register

000007BC LDI R31,0xFE Load immediate

000007BD MOV R13,R31 Copy register

000007BE MOV R31,R0 Copy register

000007BF MOVW R30,R28 Copy register pair

559: if (USART\_data\_ready())

000007C0 ADIW R30,0x01 Add immediate to word

000007C1 MOVW R14,R30 Copy register pair

000007C2 LDS R24,0x0234 Load direct from data space

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

000007C4 TST R24 Test for Zero or Minus

561: char readBuf[BUF\_SZ] = { 0 }, writeBuf[BUF\_SZ] = { 0 };

000007C5 BRNE PC+0x02 Branch if not equal

000007C6 RJMP PC+0x0047 Relative jump

000007C7 MOVW R30,R16 Copy register pair

000007C8 MOV R18,R13 Copy register

000007C9 ST Z+,R1 Store indirect and postincrement

000007CA DEC R18 Decrement

000007CB BRNE PC-0x02 Branch if not equal

000007CC MOVW R30,R14 Copy register pair

000007CD MOV R18,R13 Copy register

562: USART\_read\_data(readBuf);

000007CE ST Z+,R1 Store indirect and postincrement

000007CF DEC R18 Decrement

000007D0 BRNE PC-0x02 Branch if not equal

566: if (opcode > '8' || opcode < '0')

000007D1 MOVW R24,R16 Copy register pair

000007D2 RCALL PC-0x0598 Relative call subroutine

000007D3 MOVW R30,R16 Copy register pair

000007D4 LDD R24,Z+0 Load indirect with displacement

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

000007D5 SUBI R24,0x30 Subtract immediate

567: strcpy(writeBuf, "00Operation code is invalid");

000007D6 CPI R24,0x09 Compare with immediate

000007D7 BRCS PC+0x0A Branch if carry set

000007D8 LDI R24,0x1C Load immediate

000007D9 LDI R30,0xD6 Load immediate

000007DA LDI R31,0x01 Load immediate

000007DB MOVW R26,R14 Copy register pair

000007DC LD R0,Z+ Load indirect and postincrement

000007DD ST X+,R0 Store indirect and postincrement

000007DE DEC R24 Decrement

571: if (CommandTable[opcode].argc != readBuf[1])

000007DF BRNE PC-0x03 Branch if not equal

000007E0 RJMP PC+0x002A Relative jump

000007E1 LDI R25,0x00 Load immediate

000007E2 MOVW R30,R24 Copy register pair

000007E3 LSL R30 Logical Shift Left

000007E4 ROL R31 Rotate Left Through Carry

000007E5 ADD R30,R24 Add without carry

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

000007E6 ADC R31,R25 Add with carry

000007E7 SUBI R30,0xA0 Subtract immediate

000007E8 SBCI R31,0xFF Subtract immediate with carry

000007E9 LDD R20,Z+2 Load indirect with displacement

000007EA LDI R21,0x00 Load immediate

000007EB MOVW R30,R16 Copy register pair

000007EC LDD R18,Z+1 Load indirect with displacement

000007ED MOV R0,R18 Copy register

000007EE LSL R0 Logical Shift Left

000007EF SBC R19,R19 Subtract with carry

000007F0 CP R20,R18 Compare

572: strcpy(writeBuf, "00This function has another count of arguments");

000007F1 CPC R21,R19 Compare with carry

000007F2 BREQ PC+0x0A Branch if equal

000007F3 LDI R24,0x2F Load immediate

000007F4 LDI R30,0xF2 Load immediate

000007F5 LDI R31,0x01 Load immediate

000007F6 MOVW R26,R14 Copy register pair

--- C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

000007F7 LD R0,Z+ Load indirect and postincrement

000007F8 ST X+,R0 Store indirect and postincrement

000007F9 DEC R24 Decrement

574: CommandTable[opcode].handler(readBuf, writeBuf);

000007FA BRNE PC-0x03 Branch if not equal

000007FB RJMP PC+0x000F Relative jump

000007FC MOVW R30,R24 Copy register pair

000007FD LSL R30 Logical Shift Left

000007FE ROL R31 Rotate Left Through Carry

000007FF ADD R24,R30 Add without carry

00000800 ADC R25,R31 Add with carry

00000801 MOVW R30,R24 Copy register pair

00000802 SUBI R30,0xA0 Subtract immediate

00000803 SBCI R31,0xFF Subtract immediate with carry

00000804 LD R0,Z+ Load indirect and postincrement

00000805 LDD R31,Z+0 Load indirect with displacement

00000806 MOV R30,R0 Copy register

577: USART\_write\_data(writeBuf);

00000807 MOVW R22,R14 Copy register pair

---C:\Users\Nick\Documents\Atmel Studio\7.0\MEGA\_GPIO\_EXAMPLE2\MEGA\_GPIO\_EXAMPLE2\Debug/../src/mega\_gpio\_example.c

00000808 MOVW R24,R16 Copy register pair

00000809 ICALL Indirect call to (Z)

0000080A MOVW R24,R14 Copy register pair

---c:\program files (x86)\atmel\studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\util/delay.h

187: \_\_builtin\_avr\_delay\_cycles(\_\_ticks\_dc);

0000080B RCALL PC-0x0552 Relative call subroutine

0000080C RJMP PC-0x004A Relative jump

0000080D SER R31 Set Register

0000080E LDI R18,0xE1 Load immediate

0000080F LDI R24,0x04 Load immediate

00000810 SUBI R31,0x01 Subtract immediate

00000811 SBCI R18,0x00 Subtract immediate with carry

00000812 SBCI R24,0x00 Subtract immediate with carry

00000813 BRNE PC-0x03 Branch if not equal

00000814 RJMP PC+0x0001 Relative jump

# Приложение 3 Комментированный листинг программы для ПЭВМ на языке Си

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#include <time.h>

#include <stdlib.h>

/\*

// n = 30353, e = 185, d = 18317

1) Setup Com properties: baud, bits ...

2) use command string when using program

Debug:

n e d

add: COM4 add 16 32 64

del: COM4 del 16

set: COM4 set 16

stats: COM4 stats

encrypt: COM4 enc in out

decrypt: COM4 dec in out

\*/

#define BUF\_SZ 254

#define F\_CPU 8000000

typedef struct stats

{

uint32\_t magic;

uint32\_t nEncrypt;

uint32\_t nDecrypt;

uint32\_t nRequestNoOpenKey;

uint32\_t nEncryptedBytes;

uint32\_t nDecryptedBytes;

} STATS;

typedef struct nde

{

uint32\_t magic;

uint32\_t n; // module

uint32\_t e; // open key

uint32\_t d; // secret key

} NDE;

HANDLE ComPort = NULL;

void InitComPort(char \*com\_name)

{

DCB PORT\_dcb;

ComPort = CreateFile(com\_name, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

printf("\nPort %s is opened\n", com\_name);

// Set COM config

PORT\_dcb.BaudRate = CBR\_9600;

PORT\_dcb.StopBits = ONESTOPBIT;

PORT\_dcb.fParity = NOPARITY;

PORT\_dcb.ByteSize = 8;

printf("Baud = %d\n\n", CBR\_9600);

SetCommState(ComPort, &PORT\_dcb);

}

void RequestResponse(char str[BUF\_SZ], char rstr[BUF\_SZ])

{

DWORD bytes, bytes\_w;

WriteFile(ComPort, str, BUF\_SZ, &bytes\_w, NULL);

ReadFile(ComPort, rstr, BUF\_SZ, &bytes, NULL);

Sleep(1);

}

void EEPROM\_clear()

{

char str[BUF\_SZ] = { 0 };

char rstr[BUF\_SZ] = { 0 };

memset(str, 0, BUF\_SZ);

memset(rstr, 0, BUF\_SZ);

str[0] = '8';

str[1] = '0';

RequestResponse(str, rstr);

if (str[0] == '0')

printf("Failure: %s\n", rstr + 2);

else

printf("Success: %s\n", rstr + 2);

}

void EEPROM\_keys()

{

char str[BUF\_SZ] = { 0 };

char rstr[BUF\_SZ] = { 0 };

memset(str, 0, BUF\_SZ);

memset(rstr, 0, BUF\_SZ);

str[0] = '7';

str[1] = '0';

RequestResponse(str, rstr);

if (str[0] == '0' && str[1] == '0')

printf("Failure: %s\n", rstr + 2);

else

{

printf("Available open keys in EEPROM:\n");

uint32\_t \*p = (uint32\_t \*)rstr;

for (p = (uint32\_t \*)rstr; \*p != 0 && (char \*)p - rstr < BUF\_SZ; p++)

printf("%d) { e = %u }\n", 1 + ((char \*)p - rstr) / sizeof(uint32\_t), \*p);

}

printf("\n");

}

// show statistics recorded in EEPROM

void ShowCryptStats()

{

char str[BUF\_SZ] = { 0 };

char rstr[BUF\_SZ] = { 0 };

str[0] = '5';

str[1] = '0';

RequestResponse(str, rstr);

struct stats\* s = (struct stats\*)(rstr + 2);

printf("Statistics:\n\n"

"Number of encryption operations = %d\n"

"Number of decryption operations = %d\n"

"Number of requests without open key usage = %d\n"

"Number of encrypted bytes = %d\n"

"Number of decrypted bytes = %d\n",

s->nEncrypt, s->nDecrypt, s->nRequestNoOpenKey,

s->nEncryptedBytes, s->nDecryptedBytes);

}

// opcode - operation code, argc - number of arguments, n e d - rsa, enc - mode encrypt/decrypt used in CryptPrepare

uint8\_t CryptParams(int opcode, int argc, char \*n = 0, char\* e = 0, char \*d = 0, char enc = '0')

{

char str[BUF\_SZ] = { 0 };

char rstr[BUF\_SZ] = { 0 };

struct nde Nde;

Nde.magic = 0;

Nde.d = (d == 0) ? 0 : atoi(d);

Nde.e = (e == 0) ? 0 : atoi(e);

Nde.n = (n == 0) ? 0 : atoi(n);

str[0] = opcode;

str[1] = argc;

if (d == 0 && n == 0 && e == 0)

str[2] = enc;

else

\*(struct nde\*)&str[2] = Nde;

RequestResponse(str, rstr);

uint8\_t \*err = (uint8\_t\*)rstr;

if (\*err == '1')

printf("Success: %s\n", rstr + 2);

else

printf("Failure: %s\n", rstr + 2);

return \*err - '0';

}

void usage(char \*s = 0)

{

if (s)

printf("Unknown command '%s'\n", s);

printf("Available commands:\n");

printf("1) To add parameters n, e, d enter 'add <n> <e> <d>'\n");

printf("2) To delete parameters n, e, d enter 'del <e>'\n");

printf("3) To set default parameters n, e, d enter 'set <e>'\n");

printf("4) To show statistics enter 'stats'\n");

printf("5) To clear all EEPROM enter 'clear'\n");

printf("6) To see the list of public keys 'keys'\n");

printf("7) To generate RSA parameters enter 'gen'\n");

printf("8) To encrypt text enter 'encrypt <filename in> <filename out>'\n");

printf("9) To decrypt text enter 'decrypt <filename in> <filename out>'\n\n");

}

/////////////// RSA keys generation .. not successfull as expected

#pragma region Gen\_rsa

uint32\_t powmod(uint32\_t base, uint32\_t exp, uint32\_t mod)

{

uint32\_t res = 1;

while (exp != 0)

{

if ((exp & 1) != 0)

{

res = (1u \* res \* base) % mod;

}

base = (1u \* base \* base) % mod;

exp >>= 1;

}

return res;

}

int gcdex(int a, int b, int & x, int & y) {

if (a == 0) {

x = 0; y = 1;

return b;

}

int x1, y1;

int d = gcdex(b%a, a, x1, y1);

x = y1 - (b / a) \* x1;

y = x1;

return d;

}

int powmod\_reverse(int a, int m)

{

int x, y;

int g = gcdex(a, m, x, y);

if (g != 1)

return 0;

else {

x = (x % m + m) % m;

return x;

}

}

int Prime(unsigned a)

{

unsigned long i;

if (a == 2)

return 1;

if (a == 0 || a == 1 || a % 2 == 0)

return 0;

for (i = 3; i\*i <= a && a % i; i += 2);

return i\*i > a;

}

int gcd(int a, int b)

{

while (a != 0 && b != 0)

{

if (a > b)

a = a % b;

else

b = b % a;

}

return a + b;

}

void generate\_rsa()

{

srand(time(0));

int p = 100 + rand() % 300;

int q = 100 + rand() % 300;

int flag = 0;

int n, e, d;

while (flag != 1 || p \* q < 255 || p \* q > (uint16\_t)-1)

{

flag = 1;

while (!Prime(p)) p++;

while (!Prime(q)) q++;

int fiN = (p - 1) \* (q - 1);

e = (p + q) / 2;

while (gcd(e, fiN) != 1) e++;

d = powmod\_reverse(e, fiN);

n = p \* q;

//printf("p = %d, q = %d, n = %d, e = %u, d = %u\n", p, q, n, e, d);

for (int i = 0; i < n; i++)

{

unsigned int c = i;

unsigned int a = powmod(c, e, n);

unsigned int b = powmod(a, d, n);

if (b != c)

{

p = 100 + rand() % 400;

q = 100 + rand() % 400;

flag = 0;

break;

}

}

}

printf("p = %d, q = %d, n = %d, e = %u, d = %u\n", p, q, n, e, d);

}

#pragma endregion Gen\_rsa

// encryption: uint8\_t -> uint16\_t done by rsa algorithm

void EncryptText(char in[BUF\_SZ], char out[BUF\_SZ], uint8\_t mode)

{

FILE \*fin = fopen(in, "rb");

if (fin == 0)

{

printf("File %s not found\n", in);

return;

}

FILE \*fout = fopen(out, "wb");

int offs = sizeof(uint8\_t) + sizeof(uint8\_t) + sizeof(uint32\_t);

char buf[BUF\_SZ];

char cbuf[BUF\_SZ];

buf[0] = '0';

buf[1] = '2';

size\_t read = fread(buf + offs, sizeof(char), (BUF\_SZ - offs) / 2, fin);

while (read != 0)

{

\*(uint32\_t\*)(buf + 2) = read;

RequestResponse(buf, cbuf);

fwrite(cbuf + offs, sizeof(char), 2 \* read, fout);

read = fread(buf + offs, sizeof(char), (BUF\_SZ - offs) / 2, fin);

}

printf("Encryption from file '%s' to file '%s' is finished\n", in, out);

fclose(fin);

fclose(fout);

}

// decryption: uint16\_t -> uint8\_t done by rsa algorithm

void DecryptText(char in[BUF\_SZ], char out[BUF\_SZ], uint8\_t mode)

{

FILE \*fin = fopen(in, "rb");

if (fin == 0)

{

printf("File %s not found\n", in);

return;

}

FILE \*fout = fopen(out, "wb");

int offs = sizeof(uint8\_t) + sizeof(uint8\_t) + sizeof(uint32\_t);

char buf[BUF\_SZ];

char cbuf[BUF\_SZ];

buf[0] = '1';

buf[1] = '2';

size\_t read = fread(buf + offs, sizeof(char), (BUF\_SZ - offs), fin);

while (read != 0)

{

\*(uint32\_t\*)(buf + 2) = read / 2;

RequestResponse(buf, cbuf);

fwrite(cbuf + offs, sizeof(char), read / 2, fout);

read = fread(buf + offs, sizeof(char), (BUF\_SZ - offs), fin);

}

printf("Decryption from file '%s' to file '%s' is finished\n", in, out);

fclose(fin);

fclose(fout);

}

int main(int argc, char\*\* argv)

{

if (argc > 2 && !strncmp("COM", argv[1], 3))

{

InitComPort(argv[1]);

printf("Operation '%s'\n", argv[2]);

}

else

{

usage();

return 1;

}

// file path - 1, com port name - 2, com name - 3 args - 4+

switch (argc)

{

// 1 arg

case 3:

{

if (!strcmp(argv[2], "stats"))

ShowCryptStats();

else if (!strcmp(argv[2], "gen"))

generate\_rsa();

else if (!strcmp(argv[2], "clear"))

EEPROM\_clear();

else if (!strcmp(argv[2], "keys"))

EEPROM\_keys();

else

usage(argv[2]);

break;

}

// SetCryptParameters or DelCryptParameters

case 4:

{

if (!strcmp(argv[2], "set"))

CryptParams('4', '1', 0, argv[3]);

else if (!strcmp(argv[2], "del"))

CryptParams('3', '1', 0, argv[3]);

else

usage(argv[2]);

break;

}

// DecryptText or EncryptText

case 5:

{

if (!strcmp("encrypt", argv[2]) || !strcmp("enc", argv[2]))

{

if (CryptParams('6', '1'))

EncryptText(argv[3], argv[4], 0);

}

else if (!strcmp("decrypt", argv[2]) || !strcmp("dec", argv[2]))

{

if (CryptParams('6', '1', 0, 0, 0, '1'))

DecryptText(argv[3], argv[4], 1);

}

else

usage(argv[2]);

break;

}

// AddCryptParameters

case 6:

{

if (!strcmp(argv[2], "add"))

CryptParams('2', '3', argv[3], argv[4], argv[5]);

else

usage(argv[2]);

break;

}

default: usage(argv[2]); break;

}

CloseHandle(ComPort);

return 0;

}