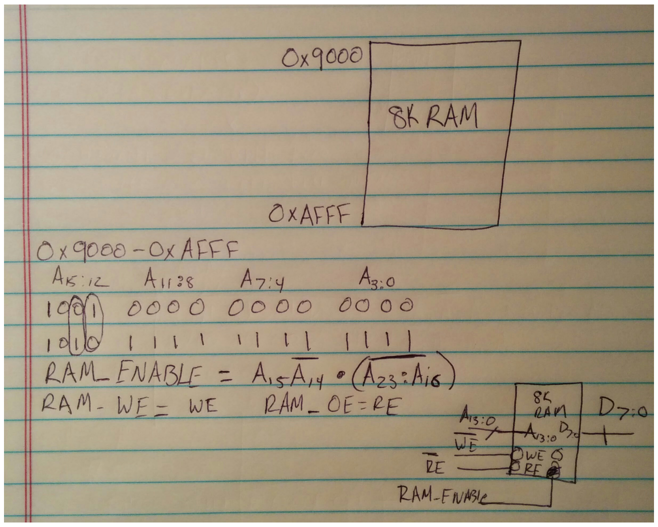
Aviles, Jean-Ralph
EEL3744
Section 1539
October 20th, 2015
Lab 4

Questions

1. What is the size of *CS1* used in this lab?
   64K - 0x470000:0x47FFFF

2. What address lines **MUST** be present in the SRAM chip-enable equation for **full address decoding**?
   A_{12} to A_{23} are required as those are the ones that are constant for all 32K addresses in the SRAM.

3. Draw a schematic of an **8**k x 8 RAM expansion starting at address 0x**9**000.
   You may only use address lines for decoding.



4. On the uPAD Proto Base board, is the CPLD required for interfacing with the SRAM? If not, explain.
   The CPLD is **not** required for interfacing with the SRAM. If we specify our Chip-select to exactly match the size of the device and we don't use any address decoding we can simply hook up the address and data lines, connect chip-enable to the chip select signal, and connect read and write enable to the signals from the XMEGA.

5. The simple memory test program described above is not very good. It checks for neighboring data pins that are shorted or left unconnected, but we have no idea if the address bus is working. Describe a procedure for testing the address lines. Are there any limitations for your procedure (for example, does your procedure test **all** of the

address lines)? Explain.

One way we could test all the addresses is to write at every memory location first the lower bytes of the address used to index a location, check to ensure they are all correct, then move on to the next byte in the address and so on. This allows us to test to see that all addresses are correctly being accessed. If there are two addresses mapped to the same data location in the RAM then when you check the program on one of the passses you'll see that the byte you wrote to that location doesn't match up with what it was expected to be. this has no limitations and tests all the address lines used to index the RAM.

6. In this lab, we configure **CS1** to be bigger than the SRAM so that we can divide it for multiple devices. If we want to connect many devices to the memory bus, this is a good way to conserve XMEGA chip select outputs. Suppose we wanted to configure **CS1** to span from 0x4**6**0000 to 0x47FFFF instead. Is this a valid range for an XMEGA chip select? With the hardware on your uPAD Proto Base as it is now, can we divide this new **CS1** the same way we do in this lab? Why or why not?

   First that range is valid $0x47FFFF - 0x460000 = 128K$ With the hardware on our Protobase as it is now we **cannot** divide this addres range since we only have access to $A_{15} : A_0$. To disambiguate all addresses in that range we would also need access to the A_{16} address line.

Problems Encountered

Lots of sloppy errors with the Quartus design and a slight mistake in the configration for the chip selects caused me quite a bit of grief. But other than that everything went well.

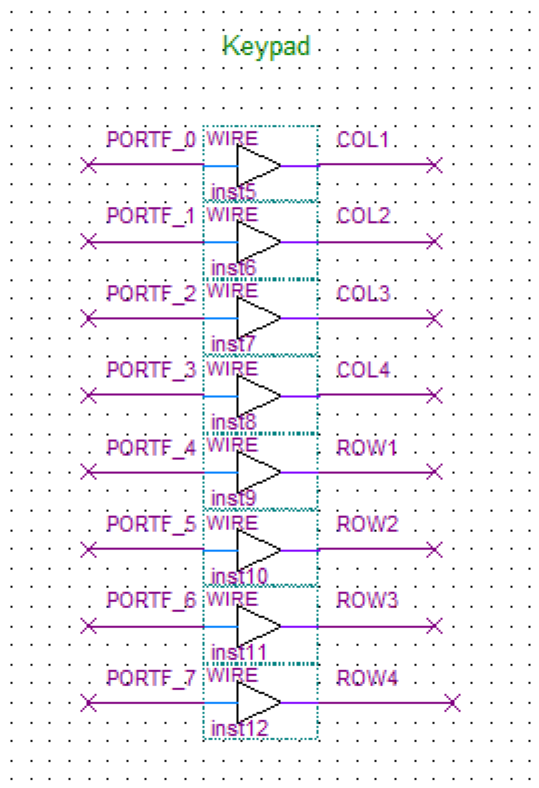
Future Work/Applications

Memory map all the things and add 16MB of external SRAM!

Appendix

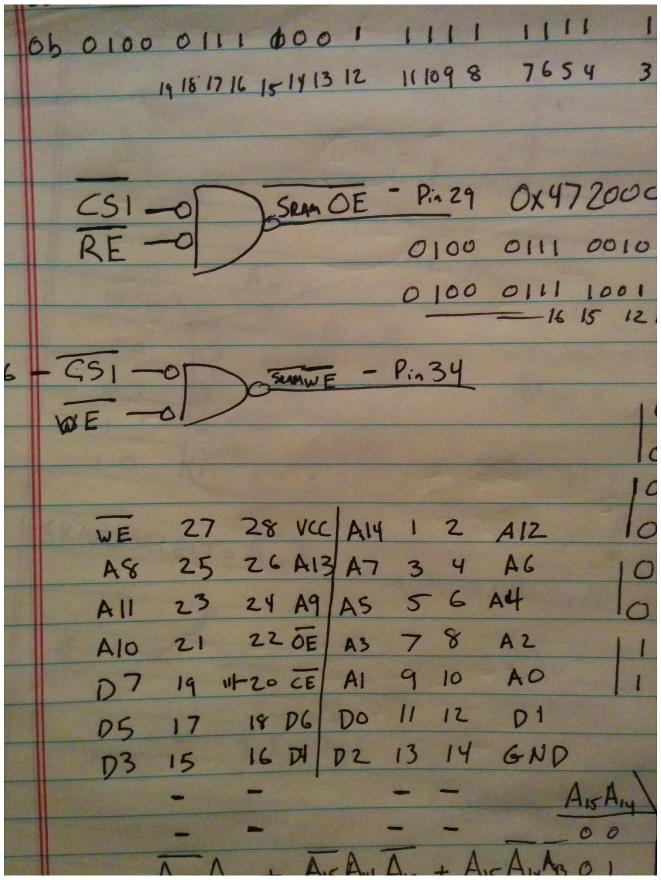
Part A

Wiring diagram for Keypad



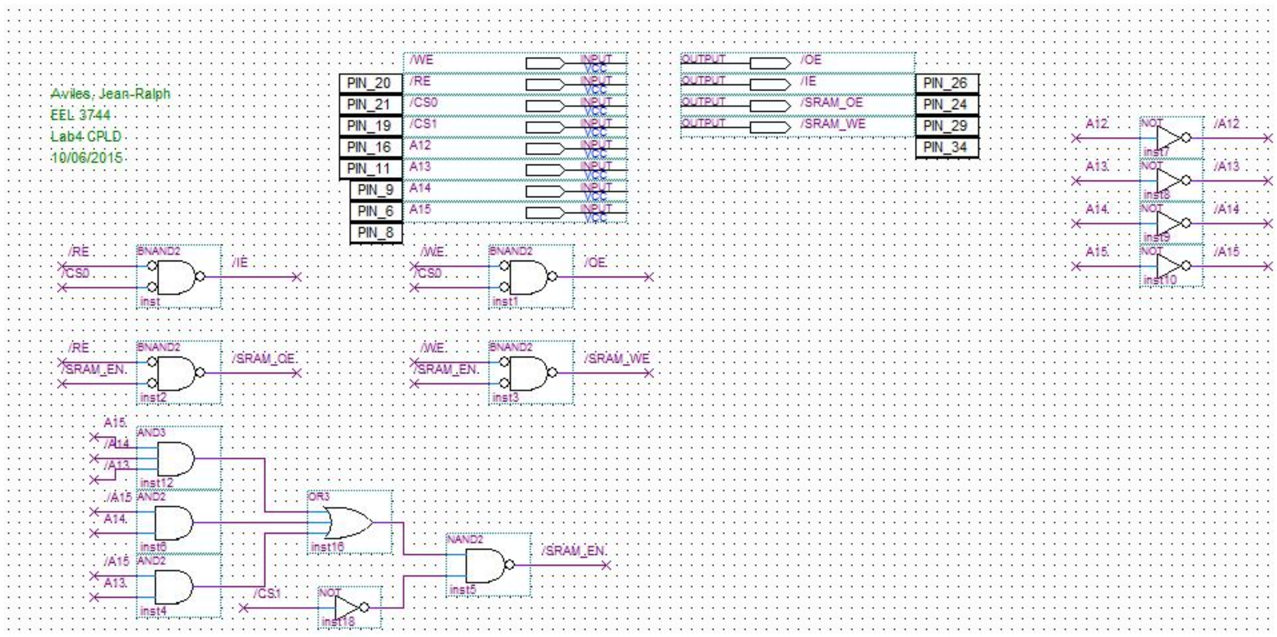
Part B

Wiring diagram for SRAM



$\overline{WE}$	27	28	VCC	A14	1	2	A12
A8	25	26	A13	A7	3	4	A6
A11	23	24	A9	A5	5	6	A4
A10	21	22	$\overline{OE}$	A3	7	8	A2
D7	19	20	$\overline{CE}$	A1	9	10	A0
D5	17	18	D6	D0	11	12	D1
D3	15	16	D4	D2	13	14	GND

Quartus Schematic



Pseudocode/Flowcharts

Part A

Pseudocode for reading from Keypad

```
# Pseudocode to read from the Keypad
def read_keypad():
    keytab = [
        [0x0D, 0x0F, 0x00, 0x0E],
        [0x0C, 0x09, 0x08, 0x07],
        [0x0B, 0x06, 0x05, 0x04],
        [0x0A, 0x03, 0x02, 0x01]
    ]
    for i in range(0, 4):
        row = (0b1000 >> i)
        output(row)
        col = read()
        for j in range(0, 4):
            in_mask = (0b1000 >> j)
            if (col & in_mask is not 0):
                return keytab[row][col]
        return 0xFF
```

Part B

Pseudocode for part B

```
# Configure Chip Selects
def main():
  write32k(0xAA)
  check32k(0xAA)
  write32k(0x55)
  check32k(0x55)

write32k(value):
  for Y in range(0x472000, 0x480000):
    *Y = value

check32k(value):
  count = 0
  X = 0x2000
  for Y in range(0x472000, 0x480000):
    tmp = *Y
    if tmp is not value:
      *(X++) = Y_L
      *(X++) = Y_H
      *(X++) = RAMPY
      count = count + 1
      if count is 100:
        break
```

Programs

Part A

Assembly for reading from Keypad

```
/*
* Jean-Ralph Aviles
* Lab 4 Keypad Reading
* 10/06/2015
* TA Khaled
*/

.include "ATxmega128A1Udef.inc"

.equ START = 0x8000     ; Start Address for IO
```

```
.equ KEYTAB_WIDTH = 4    ; Width for Keypad Table
.equ KEYTAB_HEIGHT = 4   ; Height for Keypad Table

.org 0x100
  rjmp CONFIGURE

CONFIGURE:
.org 0x200
  ldi r16, 0xF0  ; PortF pin dir, upper bits output lower input
  sts PORTF_DIR, r16  ; ""
  ldi r16, 0x0F  ; Set mask for pins to configure 3:0
  sts PORTCFG_MPCMASK, r16  ; Load mask into MPCMASK register.
  ldi r16, 0x10  ; Configure input pin pull up resistors.
  sts PORTF_PIN0CTRL, r16  ; Pin0 pullup
                           ; (Applies to 3:0 due to MPCMASK).

  ; Mem mapped I/O
  ldi R16, 0b10111  ;set /WE, /RE, /CS0 to Output
  sts PORTH_DIR, R16
  ldi R16, 0b10100  ;set /RE, /WE, /CS0 to default of 1
  sts PORTH_OUTSET, R16
  ldi R16, 0xFF  ; set all PORTJ pins (D0-D7) to be outputs.
  sts PORTJ_DIR, R16  ; As required in the data sheet.
  ldi R16, 0xFF  ; set all PORTK pins (A0-A15) to be outputs.
  sts PORTK_DIR, R16  ; As required in the data sheet.
  ldi R16, 0x01  ; Store 0x01 in EBI_CTRL register to select
                 ; 3 port EBI(H, J, K) mode and SRAM ALE1 mode
  sts EBI_CTRL, R16

  ldi r16, 0b0010101       ; Set CTRL A to 8K address size
  sts EBI_CS0_CTRLA, r16   ; (0b00101) and SRAM Mode (0b01)
                           ;  pg 335 8331
  ldi ZL, LOW(EBI_CS0_BASEADDR)  ; Load Z with BASEADDR
  ldi ZH, HIGH(EBI_CS0_BASEADDR)  ; We will load the upper 12
  ldi r16, byte2(START) ; bits of the START address
  st Z+, r16  ; into BASEADDR register.
  ldi r16, byte3(START)
  st Z, r16

  ldi r16, byte3(START)  ; Set third byte of X
  sts CPU_RAMPX, r16
  ldi XL, LOW(START)  ; Load X with START address
  ldi XH, HIGH(START)

  ldi r16, 0x0  ; Zero out the LEDs
  st X, r16
  rjmp MAIN

MAIN:
  call READ_KEYPAD  ; Load value from keypad.
  st X, r16  ; Store that value in the LEDS
  rjmp MAIN  ; Wait for another input.

READ_KEYPAD:     ; Result stored in r16
  push ZL        ; Save ZL register
  push ZH        ; Save ZH register
  push r13       ; Save r13
  push r14       ; Save r14
  push r17       ; Save r17
  ldi r16, 0x00 ; Load r16 with 0
READ_KEYPAD_1:
  cpi r16, KEYTAB_HEIGHT  ; Compare r16 with Keytab Height
  breq READ_KEYPAD_EXIT1  ; If Equal exit
  push r16  ; Save r16
  mov r17, r16  ; Save r16 into r17
  ldi r16, 0x80  ; Load r16 with output mask
  call ROTATE_RIGHT  ; Rotate r16 r17 times
  mov r17, r16  ; Move rotated value into r17
  pop r16  ; Restore r16

  sts PORTF_OUT, r17  ; Write output mask to keypad.
  nop  ; NOP Needed for some reason
  lds r13, PORTF_IN  ; Read response into r13
  ldi r17, 0x0F  ; Mask for lower bits of response.
  and r13, r17  ; Mask the response.
  ldi r17, 0  ; Load 0 into r17
READ_KEYPAD_2:
  cpi r17, KEYTAB_WIDTH  ; Compare r17 with width
  breq READ_KEYPAD_2_END  ; If equal break
  push r16  ; Else Save r16
  ldi r16, 0x08  ; Store mask into r16
  call ROTATE_RIGHT  ; Rotate response r17 times.
  mov r14, r16  ; Return value to r14
  pop r16  ; Restore r16
  cp r14, r13  ; Compare rotated mask with response.
  breq READ_KEYPAD_READ_TAB  ; If equal read value from KEYTAB.
  inc r17  ; Else decrement r17.
  rjmp READ_KEYPAD_2  ; Loop
READ_KEYPAD_READ_TAB:
  ; Read ret value from KEYTAB to r16
  ; r16 is row offset, r17 is col offset.
  ldi ZL, LOW(KEYTAB<<1)  ; Load KEYTAB into Z
  ldi ZH, HIGH(KEYTAB<<1)  ; Load KEYTAB into Z
  push r16  ; Save r16
  ldi r16, KEYTAB_WIDTH  ; Load WIDTH into r16
  mov r13, r16  ; Move WIDTH into r13
  pop r16  ; Restore r16
  mul r13, r16  ; Multiply Offset by width
  mov r16, r0  ; Get Lower value of answer.
  call INCZ_NTIMES  ; Increment Z r16 times
  mov r16, r17  ; Get COL offset
  call INCZ_NTIMES  ; Increment Z again
  lpm r16, Z        ;  Get Keypad Value
  rjmp READ_KEYPAD_EXIT
READ_KEYPAD_2_END:
  inc r16  ; Decrement r16 loop var
  rjmp READ_KEYPAD_1  ; Jump to begining of loop
READ_KEYPAD_EXIT1:
  ldi r16, 0xFF  ; Load r16 with
READ_KEYPAD_EXIT:
  ldi r17, 0x0  ; Assert 0 to the Keypad
  sts PORTF_OUT, r17  ; Write that 0
  pop r17          ; Restore r17
  pop r14          ; Restore r14
  pop r13          ; Restore r14
  pop ZH           ; Restore ZH
  pop ZL           ; Restore ZL
  ret

INCZ_NTIMES:
  push r16  ; Save r16
  push r17  ; Save r17
INCZ_NTIMES_LOOP:
  cpi r16, 0x00  ; If r16 is 0
  breq INCZ_NTIMES_END  ; Exit
  ld r17, Z+    ;  Increment Z
  dec r16  ; Decrement r16
  rjmp INCZ_NTIMES_LOOP
INCZ_NTIMES_END:
  pop r17  ; Retstore 17
  pop r16  ; Restore r16
  ret

ROTATE_RIGHT:  ; Rotate r16 r17 times.
  push r17
```

```
ROTATE_RIGHT_START:
  cpi r17, 0x00  ; Compare r17 with 0x00
  breq ROTATE_RIGHT_END  ; If 0 jump to end.
  lsr r16  ; Else rotate right r16
  dec r17  ; Decrement r17
  rjmp ROTATE_RIGHT_START  ; Loop to begining.
ROTATE_RIGHT_END:
  pop r17
  ret

KEYTAB:  ; Keypad is inverted
  .DB 0x0D, 0x0F, 0x00, 0x0E
  .DB 0x0C, 0x09, 0x08, 0x07
  .DB 0x0B, 0x06, 0x05, 0x04
  .DB 0x0A, 0x03, 0x02, 0x01
```

Part B

Assembly for part B

```
/*
* Jean-Ralph Aviles
* Lab 3 Program Sram Check
* 10/14/2015
* TA Khaled
*/

.include "ATxmega128A1Udef.inc"

.equ START = 0x8000  ;Start Address for IO
.equ SRAM_STARTA = 0x472000  ;Start Address for SRAM
.equ KEYTAB_WIDTH = 4  ;Width for Keypad Table
.equ KEYTAB_HEIGHT = 4  ;Height for Keypad Table

.org 0x100
  rjmp CONFIGURE

CONFIGURE:
.org 0x200
  ;Initialize Stack
  ldi r16, 0xFF  ;Lower Byte of Stack Pointer
  out CPU_SPL, r16  ;Set lower byte
  ldi r16, 0x3F  ;Upper byte of Stack Pointer
  out CPU_SPH, r16  ;Set Upper byte
```

```
ldi r16, 0xF0  ;PortF pin dir, upper bits output, lower input
sts PORTF_DIR, r16  ;""
ldi r16, 0x0F  ;Set mask for pins to configure 3:0
sts PORTCFG_MPCMASK, r16  ;Load mask into MPCMASK register.
ldi r16, 0x10  ;Configure input pin pull up resistors.
sts PORTF_PIN0CTRL, r16  ;Pin0 pullup Applies to 3:0

;Mem mapped I/O
ldi R16, 0b110111  ;set /WE, /RE, /CS0, /CS1 to Output
sts PORTH_DIR, R16
ldi R16, 0b110011  ;set /RE, /WE, /CS0, /CS1 to default of H
sts PORTH_OUTSET, R16
ldi R16, 0xFF  ;set all PORTJ pins (D0-D7) to be outputs.
sts PORTJ_DIR, R16  ;As required in the data sheet.
ldi R16, 0xFF  ;set all PORTK pins (A0-A15) to be outputs.
sts PORTK_DIR, R16  ;As required in the data sheet.
ldi R16, 0x01  ;Store 0x01 in EBI_CTRL register to select 3
               ;port EBI(H, J, K)  mode and SRAM ALE1 mode
sts EBI_CTRL, R16

;Switches and Leds /CS0
ldi r16, 0b0010101      ;Set CTRL A to 8K address size
sts EBI_CS0_CTRLA, r16  ;(0b00101) and SRAM Mode (0b01)
             ;pg 335 8331
ldi ZL, LOW(EBI_CS0_BASEADDR) ;Load Z with BASEADDR
ldi ZH, HIGH(EBI_CS0_BASEADDR)  ;We will load the upper 12
ldi r16, byte2(START)  ;bits of the START address
st Z+, r16  ;into BASEADDR register.
ldi r16, byte3(START)
st Z, r16

;32K SRAM
ldi r16, 0b0100001     ;Set CTRL A to 64K address size
sts EBI_CS3_CTRLA, r16  ;(0b01000) and SRAM Mode (0b01)
ldi ZL, LOW(EBI_CS1_BASEADDR)  ;Load Z with BASEADDR
ldi ZH, HIGH(EBI_CS1_BASEADDR)  ;We will load the upper 12
ldi r16, 0x00  ;bits of the Start address
st Z+, r16  ;into BASEADDR register.
ldi r16, 0x47
st Z, r16

ldi r16, byte3(START)  ;Set third byte of X
sts CPU_RAMPX, r16
ldi XL, LOW(START)  ;Load X with START address
ldi XH, HIGH(START)
```

```
  ldi r16, 0x0  ;Zero out the LED
  st X, r16

  ldi YL, LOW(SRAM_STARTA)  ;Point Y to SRAM
  ldi YH, HIGH(SRAM_STARTA)
  ldi r16, byte3(SRAM_STARTA)
  sts CPU_RAMPY, r16
  rjmp MAIN

MAIN:
  ldi r16, 0xAA  ;Load 0xAA into r16
  rcall Write32k  ;Write32K to SRAM
  rcall Check32k  ;Check32k SRAM
  ldi r16, 0x55  ;Load 0x55 into r16
  rcall Write32k  ;Write32k to SRAM
  rcall Check32k  ;Check32K SRAM

DONE:
  rjmp DONE

Write32k:
;Writes r16 to all 32K addresses in SRAM
  push r17  ;Save r17
  push YL  ;Save YL
  push YH  ;Save YH
  lds r17, CPU_RAMPY  ;Load r17 with RampY
  push r17  ;Save RampY
  ldi YL, byte1(SRAM_STARTA)  ;PointY to SRAM
  ldi Yh, byte2(SRAM_STARTA)
  ldi r17, byte3(SRAM_STARTA)
  sts CPU_RAMPY, r17
Write32kLOOP:
  cpi YH, 0xA0 ;Compare YH with 0xA0
  breq Write32kLOOPEND  ;If equal, memory is done
  st Y+, r16  ;Else store r16 at Y and increment
  rjmp Write32kLOOP  ;Loop
Write32kLOOPEND:
  pop r17    ;Restore RAMPY
  sts CPU_RAMPY, r17
  pop YH  ;Restore YH
  pop YL  ;Restore YL
  pop r17  ;Restore r17
  ret
```

```
Check32k:
;Checks r16 to all 32K addresses in SRAM
  push r18  ;Save r18
  push r17  ;Save r17
  push YL  ;Save YL
  push YH  ;Save YH
  lds r17, CPU_RAMPY  ;Load r17 with RampY
  push r17  ;Save RampY
  push XL  ;Save XL
  push XH  ;Save XH
  lds r17, CPU_RAMPX  ;Load r17 with RampX
  push r17  ;Save RampX

  ldi XL, byte1(0x2000)  ;PointX to 0x2000
  ldi XH, byte2(0x2000)
  ldi r17, byte3(0x2000)
  sts CPU_RAMPX, r17

  ldi YL, byte1(SRAM_STARTA)  ;PointY to SRAM
  ldi Yh, byte2(SRAM_STARTA)
  ldi r17, byte3(SRAM_STARTA)
  sts CPU_RAMPY, r17
  ldi r18, 0x0  ;Set r13 to 0
Check32kLOOP:
  cpi YH, 0xA0  ;Compare YH with 0xA0
  breq Check32kLOOPEND  ;If equal, memory is done
  ld r17, Y+  ;Retrieve data at Y
  cp r16, r17  ;Compare r17 with r16
  brne Check32kLOOPERROR  ;If != Write Error
  rjmp Check32kLOOP  ;Loop
CHECK32kLOOPERROR:
  st X+, YL  ;Store Lower Address of Y
  st X+, YH  ;Store Higher Address of Y
  inc r18  ;Increment r18
  cpi r18, 100  ;Compare r18 with 100
  breq Check32kLOOPEND  ;If equal then exit
  rjmp Check32kLoop  ;Else loop
Check32kLOOPEND:
  pop r17  ;Restore RampX
  sts CPU_RAMPX, r17
  pop XH  ;Restore XH
  pop XL  ;Restore XL
  pop r17  ;Restore RAMPY
  sts CPU_RAMPY, r17
  pop YH  ;Restore YH
  pop YL   ;Restore YL
  pop r17   ;Restore r17
  pop r18   ;Restore r18
  ret
```