Introduction to Microprocessors | Embedded Systems Development

EEE 347 | CNG 336

LAB MODULE #2:

ADVANCED ASSEMBLY PROGRAMMING of EMBEDDED SYSTEMS using SUBROUTINES, STACKS and EXPANDED MEMORY

Submitted by:

Talal Shafei 2542371

Noor Ul Zain 2528644

Declaration

"The content of the report represents the work completed by the submitting team only, and no material has been borrowed in any form."

Objective

The objective of this lab is to develop a complex embedded system with advanced assembly programming. It also aims to introduce networking concepts and protocols such as CRC.

Preliminary questions

(a)

(i) Packet = 0x00 Divisor is 110101

= 000 $\boxed{0000}$ CRC transmitted.

CRC will be generated for first three bits.

000 ÷ 110101 ⇒ Remainder will be $\boxed{00000}$

⇒ Remainder same as transmitted CRC, so no CRC error

⇒ This packet is a Reset Request command packet.

(ii) Packet = 0x5F

= 010 $\boxed{1111}$ CRC transmitted

```
           11
110101 | 010 00000
        - 110101
          01 01010
        - 1 10101
          0 [11111]  → Remainder same as transmitted CRC
                       hence no CRC error.
```

⇒ The Packet is an acknowledge command packet.

(iii) Packet = 0x6B

= 011 $\boxed{01011}$ CRC transmitted

$$
\begin{array}{r}
1 \\
110101 \,\overline{)\, 01100000} \\
-\ 110101 \\
\hline
00\boxed{01010}
\end{array}
$$

Remainder is 01010 $\neq$ transmitted CRC

$\Rightarrow$ hence there is a CRC error.

(iv) Packet = 0xA6 followed by Packet = 0x25

0xA6 = 1010 0110

0x25 = 001$\boxed{0\ 0101}$ CRC transmitted.

CRC will be calculated for 1010 0110 001 } first 11 bits.

```
          1 1010011011
         ____________________
110101 | 1010 0110 001 00000
        - 110101
          0111001
        - 110101
            0011000 0
          - 110101
              000101010
            - 110101
                0111110
              - 110101
                  00101100
                    110101
                    0110010
                  - 110101
                    0[00111]
```

Remainder is 00111 ≠ transmitted CRC

⇒ there's a CRC error.

(v) Packet = 0xF2 followed by Packet = 0x26

0xF2 = 1111 0010

0x26 = 001$\boxed{0\ 0110}$ CRC transmitted.

```
         10111111110
110101 ⟌ 1111 0010 001 00000
       - 110101
         00100110
        - 110101
          0100110
        - 110101
           0100110
         - 110101
            0100111
             110101
             0100100
              110101
              0100010
               110101
               0101110
                110101
                0110110
                 110101
                 00[00110]
```

Remainder is 00110

= CRC transmitted.

There is NO error.

⇒ The first packet is a data packet, with a battery level reading of 18. The second packet is a command packet for 'log request' of previous packet.

b) Find information about following types of random-access memories, *define them in 1 or 2 sentences, and comment on their application area. Stress distinctive features*

i. ROM

Read-only-memory. It is a non-volatile type of memory such that the data is not lost if electricity supply is cut, it is used for permanent storage typically to store data that won't change with time like the booting program in the computers and in Embedded systems to store the instructions.

ii. SRAM

Static random-access memory is a volatile memory. It is fast and is used in cache and registers.

iii. DRAM

It is slower and requires periodic refreshing. It is a high-capacity version of RAM, used in the main memory of the computer.

iv. SDRAM

Synchronous DRAM or DRAM with a clock, thus faster and used in main memory for personal computers and servers.

v. DDR3 SDRAM

It has double transfer rate because of its ability to transmit data on both edges of the clock which make it have higher transfer rate for data, it is typically used graphic processing units (GPUs).

vi. FLASH

It's a form of EEPROM. It can be erased electrically and its contents can be deleted in a flash, it is much cheaper than the above and it is used in SDD as a secondary storage.

vii. *What type of RAM (pick from i-vi) is 6116? Explain*

6116 RAM is a SRAM since it does not require any clock and is asynchronous.

c) Carefully explain why the 8-bit latch-based register in Figure 2.4 is needed in ATMega128 external memory interface? What would happen if this latch were excluded from the design?

ATMega 128 will send 16-bit address and 8-bit data to the external memory, and because the lower byte of the address will be send out from the same port as the data (AD which is PORTA) they need to be send sequentially but also should arrive together to the external memory to map and write in the correct place therefore we put a latch to work as a buffer to hold the lower byte of the address

1 cycle until the ATMega128 send the data to external memory so they both arrive to the external memory at the same time. If we don't use latch the moment the address arrive the lower byte will also be the data so this will result in writing un-useful data then when data arrives next we would be already lost the lower byte of the address so we won't be able to map to the correct place in the external memory so this will result in data loss.

Code

```
;
; module_2.asm
;
; Created: 5/1/2023 6:25:38 PM
; Author : Talal Shafei and Noor Ul Zain
;


RJMP start
.INCLUDE "M128DEF.INC"

.EQU ZEROS = 0X00
.EQU ONES = 0XFF
.EQU POLYG = 0b11010100
.EQU IMEM_START = 0x121; to start after the stack
.EQU XMEM_END = 0x18FF ; 0x10FF + 800(2KB) :
; 0x7FF 0111 1111 1111  the first 8 bits will be sent through
; porta and the other 3 bits will be sent through portc[0:2]

.EQU PACKET_IN = PINB
.EQU PACKET_OUT = PORTD
.EQU READY_OUT = PORTE

.DEF TOS = R2 ; since TOS will only contain one packet at any time
.DEF IS_TOS_EMPTY = R30 ; a flag to see if TOS is empty (1: empty, 0:full)
; it will give warning since ZL is R30 but it in not a problem since we are not using
Z reg

.DEF FAIL_PASS = R25 ;(0: failed, 1: passed)

.DEF CAPTURED = R5


.MACRO PUSH_TOS
       CLR IS_TOS_EMPTY
       MOV TOS, @0
.ENDMACRO

.MACRO POP_TOS
       LDI IS_TOS_EMPTY, 0X01
       MOV @0 , TOS
.ENDMACRO


;CODE
.CSEG
.ORG 0X0050
start:

       ; Memory partition
```

```
        ; Initialize Stack pointer so we can use subroutines with no problem
        ; Note 0x120 for the stack to have at least 20 bytes
        LDI R16, LOW(0x120)
        OUT SPL, R16
        LDI R16, HIGH(0x120)
        OUT SPH, R16

        ; Initialize X as a pointer to the position in the Log file

        LDI XL, LOW(IMEM_START)
        LDI XH, HIGH(IMEM_START)

        ; Initialize for XMEM
        LDI R16, (1<<SRE) ; activate XMEM
        OUT MCUCR, R16

        LDI R16, (1<<XMM2)|(1<<XMM0) ; so we can release PC7 - PC3
        STS XMCRB, R16

        ; Initialize C
        ; C[3] is input for Start/Stop
        ; C[4] is input for Memory Dump
        ; C[5] is input for Last Entry
        ; C[6] is input for Recieve flag push down button
        ; C[7] is output for Ready (LED)
        ; 0b1000 0xxx -> 0x80
        LDI R16, 0x80
        OUT DDRC, R16

        LDI R16, ZEROS
        OUT DDRB, R16 ; PIN B is input for PACKET_IN

        ; Initialize outputs
        LDI R16, ONES
        OUT DDRD, R16 ; Port D is output for PACKET_OUT
        OUT DDRE, R16 ; Port E is output for READY_OUT



initialize:
        CBI PORTC, 7 ; make sure the led is off
        ; initialize
        CALL INIT

main:


        CALL SERVICE_OUT

        ; check start/stop
        SBIS PINC,3
        RJMP main ; if it stop go back to main

        SBI PORTC, 7 ; turn on Ready led

        SBIS PINC, 6 ; receive push down button
        RJMP main
wait_to_let_go_of_the_push_down_button:
        SBIC PINC, 6
        RJMP wait_to_let_go_of_the_push_down_button
```

```
        CBI PORTC, 7 ; turn off Ready led because now capturing Packet_in

        IN CAPTURED, PACKET_IN ; captture packet_in

        SBRS CAPTURED , 7 ; if the packet is data skip
        RJMP command_packet_in

        ; is stack (TOS) empty?
        SBRS IS_TOS_EMPTY, 0; if yes skip the popping
        POP_TOS R16; R16 temp to discard what came out of the TOS

        PUSH_TOS CAPTURED ; push the data packet to TOS
        JMP main


command_packet_in:
        ; TOS has data packet?
        SBRC TOS, 7 ; if it is set then it means it is a data packet
        JMP tos_has_data_packet

        ; since there is no data packet in TOS we will do crc3_check
        CALL CRC3_CHECK

        SBRS FAIL_PASS,0
        JMP fail
        ; since there is a data packet we need to do check11
        ; default behavior is pass
        MOV R19, CAPTURED
        ANDI R19,  0x60 ; mask the command input (0b0110 0000)
        CPI R19, 0x40; check if acknowledge (0b0100 0000)
        BRNE check_if_repeat

        ; it is acknowledge, then empty the stack and go back to main
        ; is stack (TOS) empty?
        SBRS IS_TOS_EMPTY,0; if yes skip the popping
        POP_TOS R16; R16 temp to discard what came out of the TOS
        JMP main

check_if_repeat:
        CPI R19, 0x60 ; check if repeat (0b0110 0000)
        BREQ it_is_repeat
        JMP main ; if it not repeat go back to main

it_is_repeat:
        ; is stack (TOS) empty?
        SBRC IS_TOS_EMPTY,0; if is not empty skip jumping directly to main
        JMP main
        ; not empty then pop into R17 and transmit
        POP_TOS R17
        CALL TRANSMIT
        JMP main


tos_has_data_packet:
        CALL CRC11_CHECK

        CPI FAIL_PASS, 0x01
        BREQ passed
        POP_TOS R16; R16 temperoray to discard TOS
        JMP fail
```

```
passed:
	; check if it is a log request
	MOV R19, CAPTURED
	ANDI R19,  0x60 ; mask the command input (0b0110 0000)
	CPI R19, 0x20; check if is log request (0b0010 0000)
	BREQ it_is_log
	JMP main

it_is_log:
	; it is log, then log the data that was in TOS
	ST X+, TOS
	CALL CHECK_MEMORY

	POP_TOS R16; to discard the value in the TOS

	LDI R17, 0x40 ; laod acknowledge
	CALL CRC3 ; generate the crc for the acknowledge
	PUSH_TOS R17 ; keep it in stack incase sensor asked to resent it
	CALL TRANSMIT

	JMP main



fail:
	CALL REPEAT_REQUEST
	JMP main




TRANSMIT:
	OUT PACKET_OUT, R17
	RET

INIT:
	LDI R17, 0X00 ; command reset request
	CALL CRC3 ; generate crc in R17
	CALL TRANSMIT ; send R17 to packet_out, made as a macro to have more
flexibility when calling and it is one line anyway
	PUSH_TOS R17 ; push the value to TOS
	RET


CHECK_MEMORY:
	; if they are equal check the lower byte else return where you left off
	LDI R16, HIGH(XMEM_END)
	CP XH, R16
	BREQ maybe_full
	RET
maybe_full:
	; if XL > MEM_END Lower byte reset it else return
	LDI R16, LOW(XMEM_END)
	SUBI R16, 20 ; because last 20 bytes are for stack
	CP R16, XL
	BRMI reset
	RET
reset:
	; reset to the beginning of internal sram again in round-robin fashion
	LDI XL, LOW(IMEM_START)
	LDI XH, HIGH(IMEM_START)
```

```
        RET


SERVICE_OUT:
        ; memory dump check
        SBIC PINC, 4 ; if memory_dump is not active skip jumping to it
        RJMP memory_dump

        ; last entry check
        SBIS PINC, 5 ; if we reach here that means memory dump is not active
        ; if last entry FLAG is set then skip and dont return now
        RET

        ; read last entry in the log file
        MOVW Y,X ; so we dont affect the memory pointer
        LD R16, -Y
        OUT READY_OUT, R16
        RET


memory_dump:
; so we dont discard all the bytes in the memory when we dump them
        MOVW Y, X ; maybe it should be MOVW R28, R26
loop_dump:

        CPI YL, LOW(IMEM_START)
        BRNE not_finished
        CPI YH, HIGH(IMEM_START)
        BREQ finished

not_finished:
        LD R16, -Y
        OUT READY_OUT, R16
        CALL DELAY
        CALL DELAY
        RJMP loop_dump

finished:
        RET



REPEAT_REQUEST:
        LDI R17, 0x60 ; Repeat Request
        CALL CRC3
        CALL TRANSMIT
        RET

DELAY:
        LDI R16, 0xFF
delay_loop:
        DEC R16
        NOP
        BRNE delay_loop
        RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;

CRC3:
        ; R17 is the param
        MOV R21, R17     ; copying the data input into R21
```

```
    ANDI R21, 0b11100000  ; bit masking as we are calculating the CRC3
    ANDI R17, 0b11100000  ; bit masking as we are calculating the CRC3


    LDI R16, POLYG   ; polynomial G with big endian ->even if little endian, we can 
just shift it, does not matter
       LDI R22, 0  ; will be used as shift counter

div:
       SBRC R21, 7 ; if the first bit is cleared, skip xor and shift
       EOR R21, R16
       SBRS R21, 7  ;if the xor result's MSB is not set, we shift
       JMP shift

shift:
       LSL R21
       INC R22      ; keep counting the shifts
       CPI R22, 3   ; 3 because we are creating the CRC code for the first 3 bits
       BREQ exit    ; if shifted 3 times, message is over; exit
       SBRS R21, 7  ; check MSB again after shift and if not set, loop
       BRNE shift


       BRNE div     ; Loop back to div

    JMP exit

exit:
       ROR R21         ;
       ROR R21         ; since the first 5 bits are CRC and I want to add it to R17 
(original)
       ROR R21         ;
       Add R17, R21    ; append CRC to input
       RET



CRC3_CHECK:
       LDI FAIL_PASS, 0x00 ; set the falg to fail
       MOV R17, CAPTURED
       MOV R20, CAPTURED

       CALL CRC3

       CP R20, R17   ; R17 is used by CRC3 and should have the appended CRC to input
       BREQ not_corrupted
       RET

not_corrupted:
       LDI FAIL_PASS, 0x01 ; if it not corrupted set the flag to pass
       RET

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;

CRC11:

       ;R17 has the Higher byte and R18 the Lower byte -> R17:R18

       MOV R21, R17         ; copying the data input into R21 -> in case original 
data is needed again
```

```
        LDI R16, POLYG ; polynomial G with big endian (even if little endian, we can
shift)

        LDI R22, 0  ; will be used as shift counter for lower byte
        LDI R23, 0  ; will always stay 0
        LDI R24, 0  ; will be used a shift counter for total shifts



div11:
        SBRC R21, 7 ; if the first bit is cleared, skip xor and shift (v imp)
        EOR R21, R16
        SBRS R21, 7  ;if the xor result's MSB is not set, we shift
        JMP shift11

shift11:
        LSL R21


        CPI R22, 3    ;as we only want 3 bits we want from lower byte
        BREQ go_here
        LSL R18
        ADC R21, R23  ;add the shifted lower byte's carry to R21
        INC R22


go_here:
        INC R24 ; keep counting the shifts
        CPI R24, 11  ;  11 because we are creating the CRC code for the first 11 bits
        BREQ exit11    ;  if shifted 11 times, message is over; exit
        SBRS R21, 7  ;  check MSB again after shift and if not set, keep shifting
        BRNE shift11

        BRNE div11    ;Loop back to div

        JMP exit11

exit11:
        ROR R21   ;
        ROR R21   ;  Rotating three times to get the CRC to be last 5 bits (back to
little endian)
        ROR R21   ;  Note that cannot use swap here 00111(CRC)000 swapped would be
10000011->incorrect

        RET



CRC11_CHECK:
        LDI FAIL_PASS, 0x00 ; set falg as failed
        MOV R17, TOS    ; the data input (highbyte)
        MOV R18, CAPTURED          ; the data input (lowbyte)

        CALL CRC11

        MOV R20, CAPTURED
        ANDI R20, 0b00011111 ; mask the lower byte to extract the last 5 bits only

        CP R20, R21                        ; check CRC11 for both these registers
```

```
        BREQ not_corrupted
        RET
not_corrupted11:
        LDI FAIL_PASS, 0x01 ; set flag as passed
        RET
```

Testing:

1. RESET Command

We will send a Reset Request and we will store it inside TOS in case we receive a Repeat Request:

INIT is called.

Inside it we load R17 with 0x00 for the Reset request

R17	0x00

then call CRC3 on R17, which should result in 0x00 too

R17	0x00

Then transmit it to packet out

```
TRANSMIT:
    OUT PACKET_OUT, R17
    RET
```

I/O PORTD	0x32	0x00	

and store it inside TOS (R2)

R02	0x00

2. Checking user inputs and system response

Here we set start to 1.

I/O PINC	0x33	0x08	

Ready led is ON

I/O PORTC	0x35	0x80	

We press the push down button for Receive and let go.

```
wait_to_let_go_of_the_push_down_button:
    SBIC PINC, 6
    RJMP wait_to_let_go_of_the_push_down_button
```

And ready led is off once we change receive to 0.

3. Sensor sends ACKNOWLEDGE command with correct CRC

Sensor will send Acknowledge Command packet In (0101 1111).

Turn off the Ready LED led

Then capture the command packet in CAPTURED (R5)

then we will check if it is a command type

```
SBRS CAPTURED , 7 ; if the packet is data skip
RJMP command_packet_in
```

Then it will check if TOS has a data packet, but it doesn't because it has the last packet_out the reset one.

```
command_packet_in:
    ; TOS has data packet?
    SBRC TOS, 7 ; if it is set then it means it is a data packet
    JMP tos_has_data_packet
```

So now we check the crc3 by calling CRC3_CHECK

It passed the check because it is having the 11111 appended to it like the one generated by the CRC3.

```
not_corrupted:
    LDI FAIL_PASS, 0x01 ; if it not corrupted set the flag to pass
    RET
```

Now since it passes, we will check if the captured values is an acknowledge command.

It is so now we will check if stack is empty.

It is not so we will pop TOS to get rid of the Reset Command that was in it since we won't need to send it back because the Sensor confirmed receiving it, finally we jump back to main.

```
POP_TOS R16; R16 temp to discard what came out of the TOS
JMP main
```

4. Repeat/ Error subroutine

We will check the Repeat/ Error subroutine by sending a corrupted Acknowledge command from the sensor so the MCU.

MCU must reply back by sending Repeat command to sensor.

Corrupted command (010 11101)

It skipped the BREQ because it is corrupted:

So, fail pass register will stay at fail: 0x00

```
    CP R20, R17    ; R17 is us
    BREQ not_corrupted
    RET
```

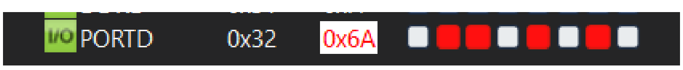
We call REPEAT REQUEST:

```
fail:
    CALL REPEAT_REQUEST
    JMP main
```

The subroutine will load 0x60 in R17.

Then generate the CRC3 which will make it 0x6A (0110 1010) -> the repeat request.

And finally, we will transmit it to PORTD.



5. Sensor sends Data Packet

Sensor will send a Data packet and MCU will save it to TOS.

Data packet is for Battery level (1111 1111)

Because we restarted here the TOS had Reset command, so we pop first then we store 0xFF in TOS (R2)

6. Sensor sends LOG command

Log request is 0x39 (0011 1001) because it is crc11 since it follows 0xFF data packet from before from before.

Like before first it will check if it is a command packet, which it is then it will check if TOS has a Data packet, and because it follows from before, TOS has a data packet which is 0xFF.

```
tos_has_data_packet:
    CALL CRC11_CHECK
```

Now it will call CRC11_CHECK to check if the 16 bit was received correctly.

```
not_corrupted:
    LDI FAIL_PASS, 0x01 ; if it not corrupted set the flag to pass
    RET
```

Now since it passed the check we need to see if it is a log request by masking the 2 bits [6:5] to see if they are 01.

```
passed:
    ; check if it is a log request
    MOV R19, CAPTURED
    ANDI R19,  0x60 ; mask the command input (0b0110 0000)
    CPI R19, 0x20; check if is log request (0b0010 0000)
    BREQ it_is_log
    JMP main
```

Indeed they are!

Now we need to log TOS to the SRAM

```
it_is_log:
    ; it is log, then log the data that was in TOS
    ST X+, TOS
    CALL CHECK_MEMORY
```

We call check memory to make sure that if we reach the last address 0x18FF (0x10FF Internal + 0x800 External) to reset the X pointer to 0x121 (because the first 20 bytes were reserved for the SP)

```
Memory: prog FLASH
data 0x0121  ff 00 00 00 00 00 00 00 00 00
```

Data was logged successfully, but now we need to transmit an acknowledge command in packet out to tell the sensor that we logged the data.

First, we pop TOS then generate CRC3 for Acknowledge Command packet, and load it in TOS and Transmit it in Packet out.

```
POP_TOS R16; to discard the value in the TOS

LDI R17, 0x40 ; laod acknowledge
CALL CRC3 ; generate the crc for the acknowledge
PUSH_TOS R17 ; keep it in stack incase sensor asked to resent it
CALL TRANSMIT

JMP main
```

Now PortD has the Acknowledge command 0x5F (0101 1111) as we saw before in test 3.

```
I/O PORTD    0x32    0x5F
```

7. Sensor sends REPEAT command:

In this test we will send a Repeat command from the sensor, and since we restarted the program, we should re-send the Reset command that we saved in TOS.

Packet in will be Repeat command 0x6A (0110 1010).

Like before we are going to check if it is a command packet. If yes, we are going to call crc3_check and then we check if it Acknowledge but this time it's not so we it will check next if it is a Repeat request.

We check if TOS is empty by checking the flag it was set to 1, then we pop the TOS into R17, and call Transmit to outputs R17 into PortD.

```
it_is_repeat:
    ; is stack (TOS) empty?
    SBRC IS_TOS_EMPTY,0; if is not empty skip jumping directly to main
    JMP main
    ; not empty then pop into R17 and transmit
    POP_TOS R17
    CALL TRANSMIT
    JMP main
```

```
I/O PORTD    0x32    0x00
```

PortD is 0x00 that means we transmitted the Reset Request again like expected.

8. Last Entry asserted with Memory Dump off

In this test we assert Last Entry to get the last logged data to be outputted in the Ready out LEDs, (PORTE), by calling SERVICE_OUT subroutine and making sure memory dump flag is off.

We already made the process to log 2 values 0xFF and 0xFE and we are supposed to see 0xFE in the Ready out.

PINC will be to 0x20 (0010 0000) to activate only the last entry bit.

As you can see below last log is 0xFE

```
data 0x0121  ff fe 00 00 00 00 00 00 00
```

Now we save the pointer X value in Y, so we don't lose it since we need to make pre-decrement before loading the value in LED (because when we stored we use post increment so X always points to empty values)

```
; read last entry in the log file
MOVW Y,X ; so we dont affect the memory pointer
LD R16, -Y
OUT READY_OUT, R16
RET
```

And finally, we the last logged value in PortE as expected

```
I/O PORTE    0x23    0xFE
```

9. Memory Dump is on

final test will be for memory dump where are we going to dump first 0xFE then 0xFF from the log file, by also calling SERVICE_OUT subroutine and this time we will execute the memory dump part.

Note here we commented out the Delay calls.

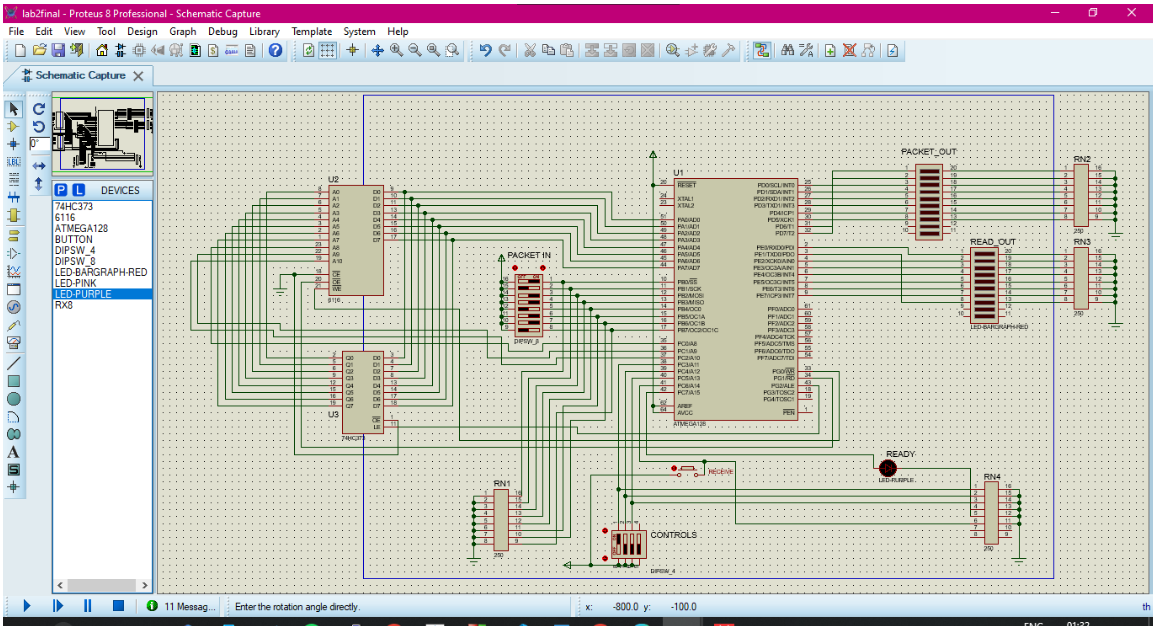
User will activate the memory dump switch: 4th bit in PINC thus we set PINC to 0x10.

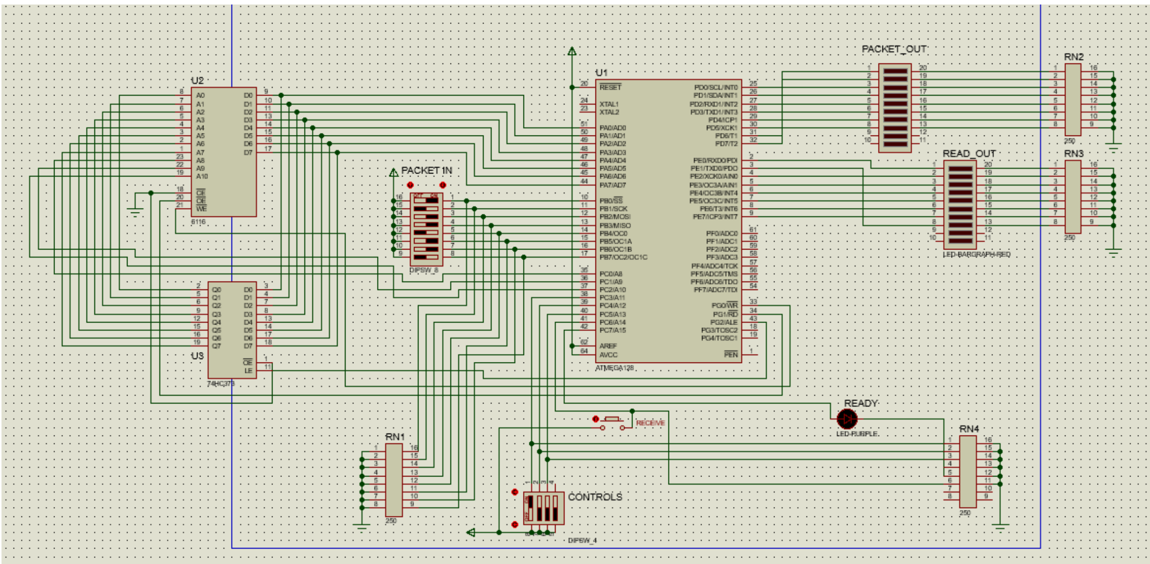
Like before we copied X pointer value to Y, so we don't lose track of our log file and then we logged the values to PORTE, and we test each time we logged that we didn't reach our starting position IMEM_START (0x121)

```
memory_dump:
; so we dont discard all the bytes in the memory when we dump them
    MOVW Y, X ; maybe it should be MOVW R28, R26
loop_dump:

    CPI YL, LOW(IMEM_START)
    BRNE not_finished
    CPI YH, HIGH(IMEM_START)
    BREQ finished

not_finished:
    LD R16, -Y
    OUT READY_OUT, R16
    ;CALL DELAY
    ;CALL DELAY
    RJMP loop_dump

finished:
    RET
```

And the values will be logged to PORTE as follow

Proteus Design



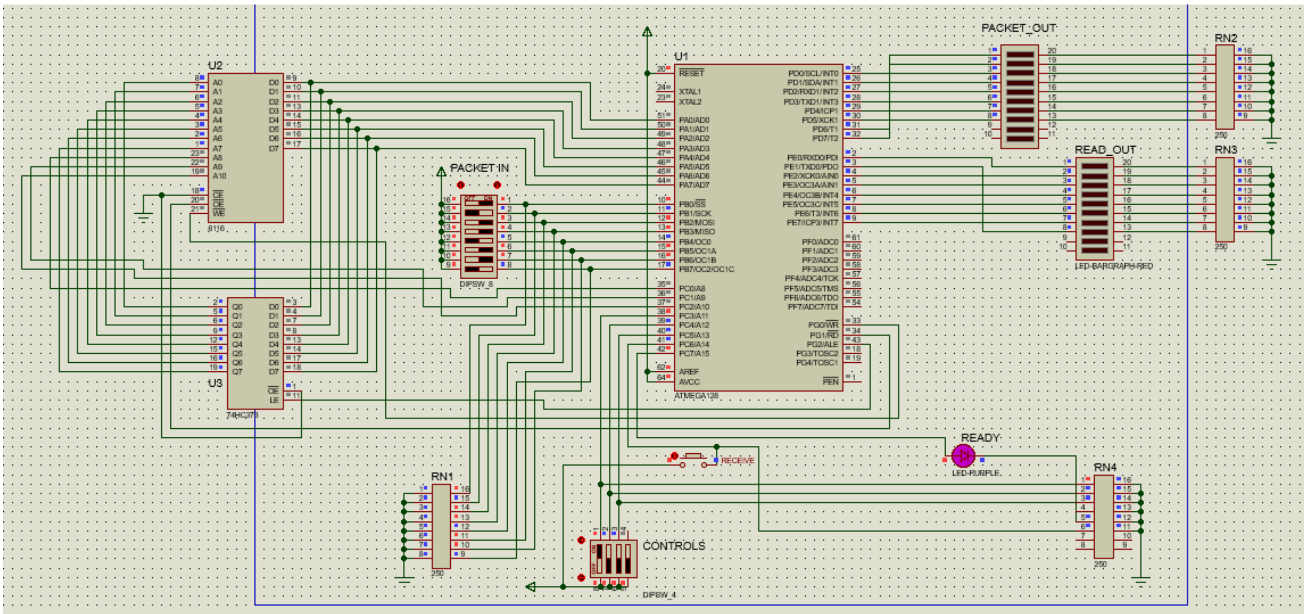


TEST CASES:

CASE 1 (a):

Reset request sent and receive is not asserted.

PACKET_IN is not captured.



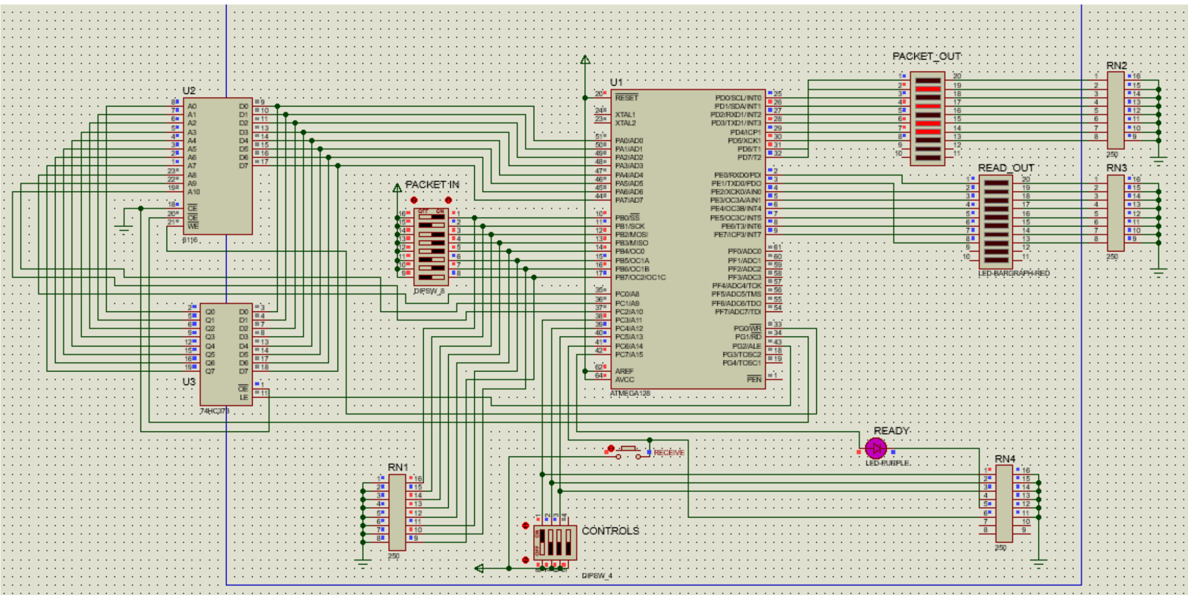
Notice that READY LED is ON and will stay ON unless Receive push button is pressed.

CASE 1 (b):

Receive is asserted. Notice that the PURPLE LED switches off indicating that the PACKET_IN has been captured.

Here PACKET_IN is an ACKNOWLEDGE PACKET with the WRONG CRC -> 010 11101 (5D). The CRC bits are highlighted.

We expect to see a REPEAT REQUEST as PACKET_OUT -> 011 01010 (6A).



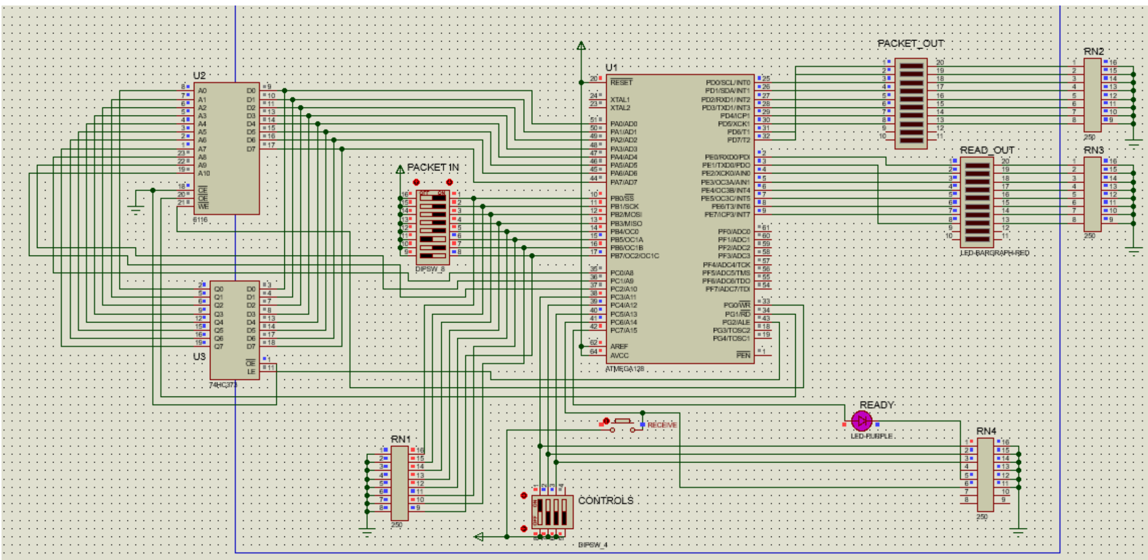
We see the correct output (REPEAT_REQUEST of 011 01010, read the LED bits from down to up starting from the 8th bit)

CASE 1 (c):

Receive is asserted.

Here PACKET_IN is an ACKNOWLEDGE PACKET with the RIGHT CRC -> 010 11111 (5F). The CRC bits are highlighted.

We expect to see NO PACKET_OUT.



CASE 2 (a):

Data packet followed by incorrect log request.

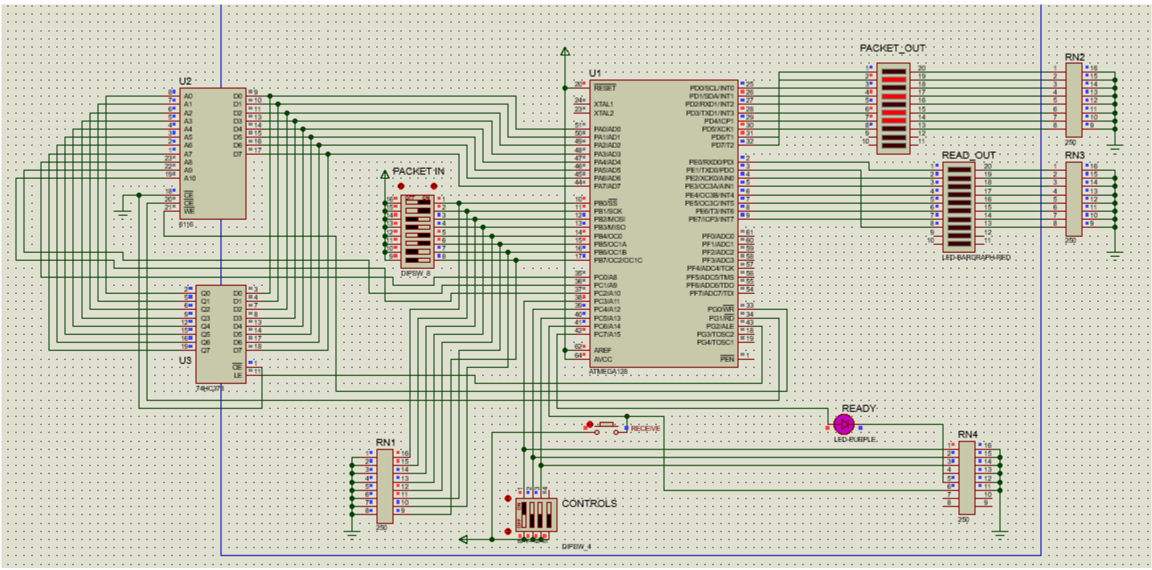
DATA PACKET_IN: 10001111 (8F)

It will be stored in TOS.

Followed by an incorrect LOG request

COMMAND PACKET_IN: 001 10011 (incorrect CRC for the 11 bits, starting from MSB of data, highlighted)

As expected, REPEAT_REQUEST's outcome, 011 01010 seen as PACKET_OUT.



CASE 2 (b):

Data packet followed by correct log request resulting in acknowledge output.

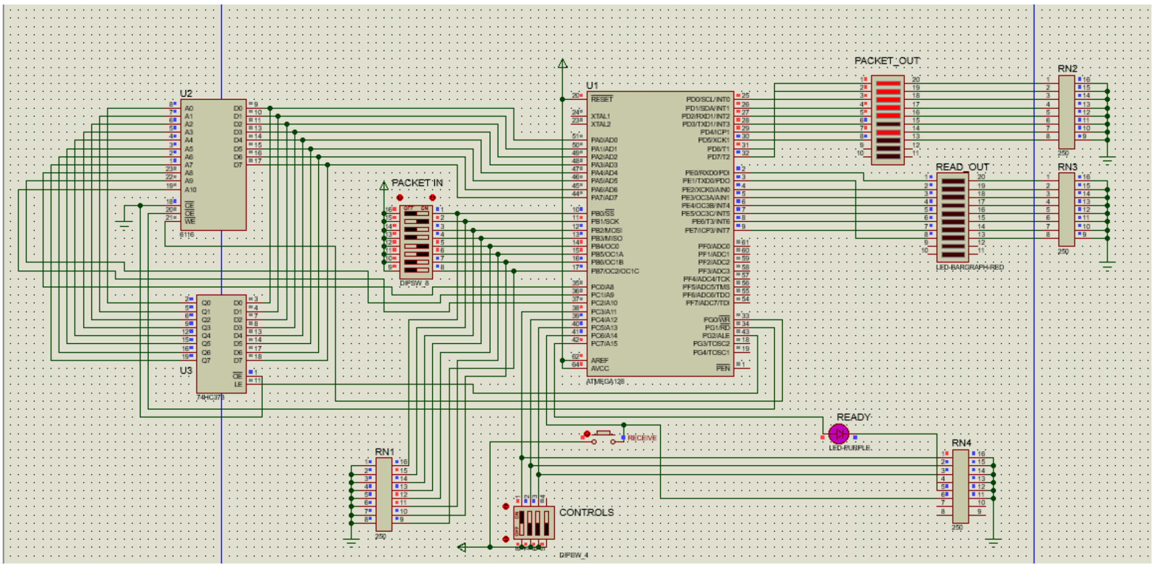
DATA PACKET_IN: 10001111 (8F)

It will be stored in TOS.

Followed by a correct LOG request.

COMMAND PACKET_IN: 001 10010 (correct CRC for the 11 bits, starting from MSB of data, highlighted)

As expected, ACKNOWLEDGE command packet (010 11111) seen as PACKET_OUT, indicating that logging was successful.



AVR Data Memory - U1

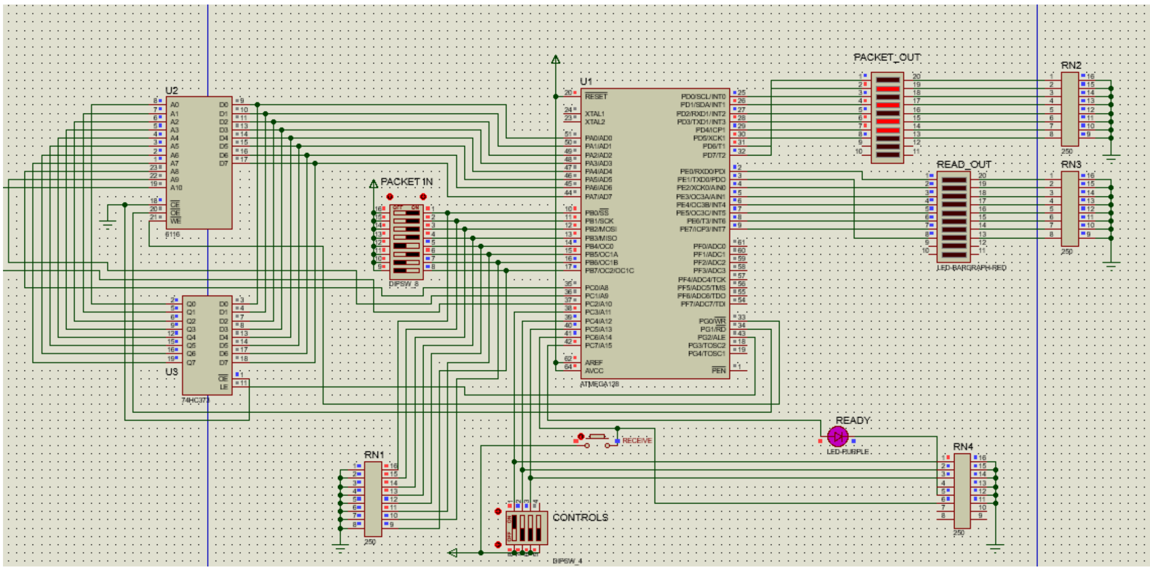
0100	00	00	00	00	00	00	00	00	00	00	00	00	00
0118	00	00	00	00	00	00	EB	00	69	8F	00	00	00
0130	00	00	00	00	00	00	00	00	00	00	00	00	00
0148	00	00	00	00	00	00	00	00	00	00	00	00	00
0160	00	00	00	00	00	00	00	00	00	00	00	00	00

AS seen, 8F is logged into AVR data memory.

CASE 3:

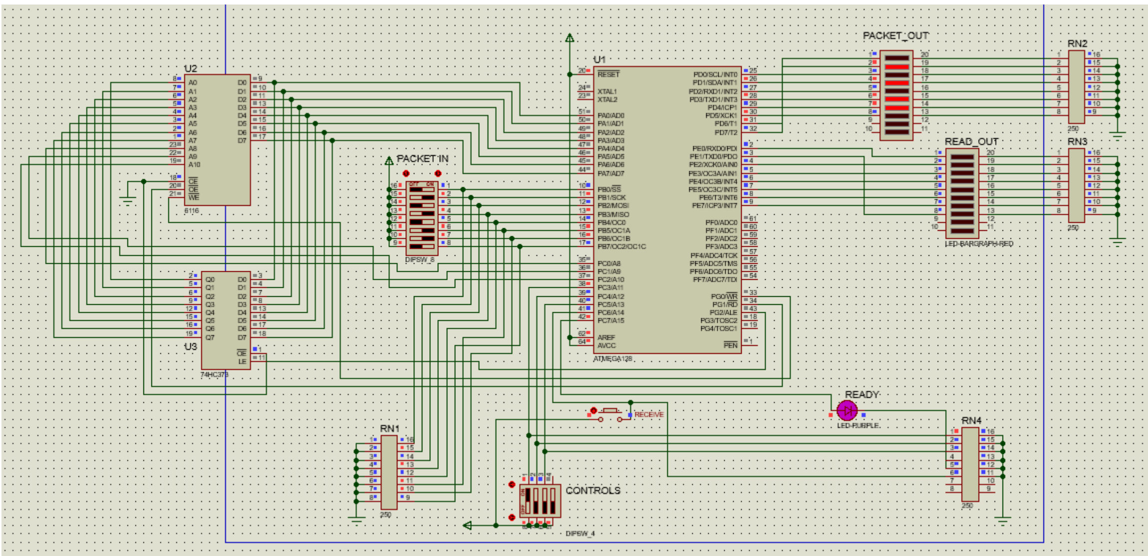
Received a REPEAT request from sensor.

Assume CASE 2(a) took place:

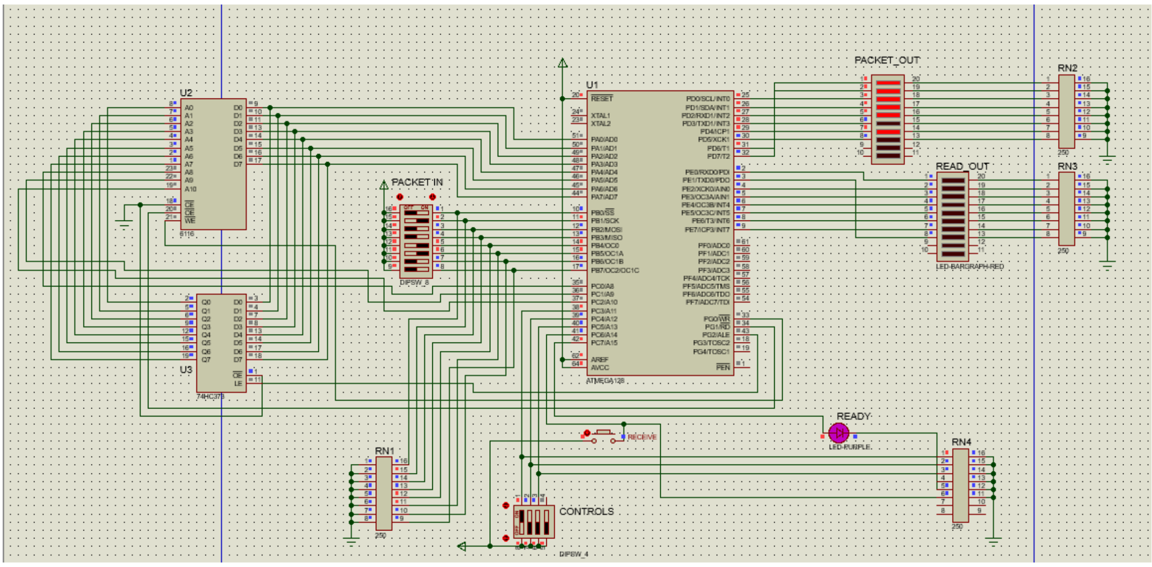


And now TOS has the error request: 011 01010 (6A).

Now, let's change the DATA PACKET_IN: 011 01010 (6A). It is also the error request received from the sensor. Let's see if we have the TOS as the packet_out. Indeed we do!!

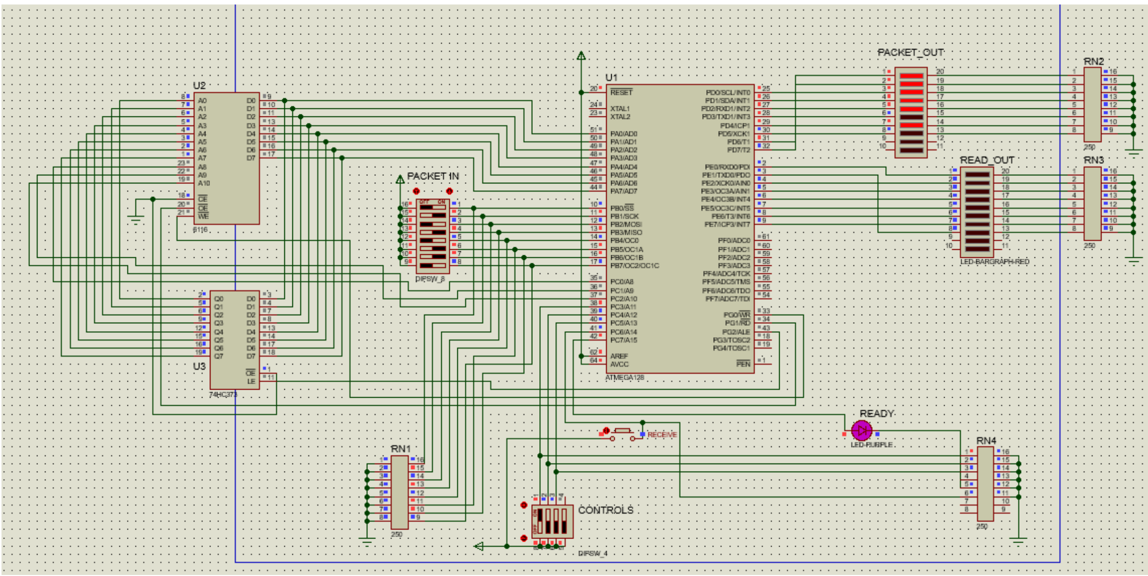


For the sake of completeness, now assume CASE2 (b) took place:



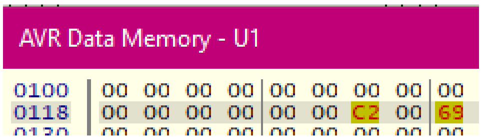
So that TOS has the Acknowledge packet (010 11111).

Now, let’s change the DATA PACKET_IN: 011 01010 (6A). It is the error request received from the sensor. Let’s see if we have the TOS as the packet_out. Indeed, we do!!

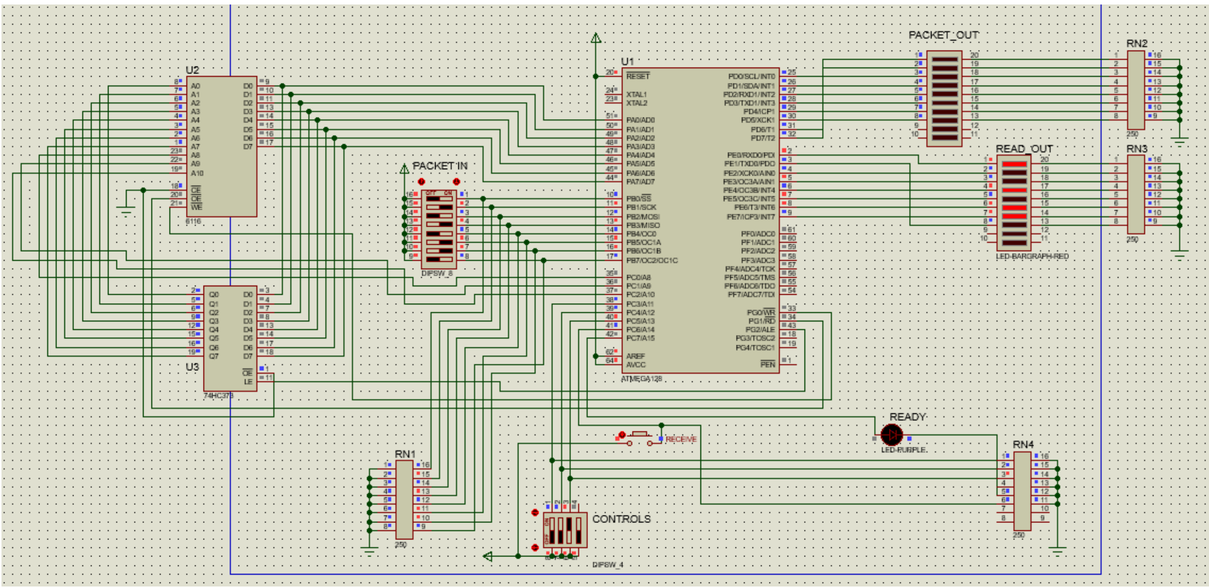


CASE 4:

LAST ENTRY asserted.



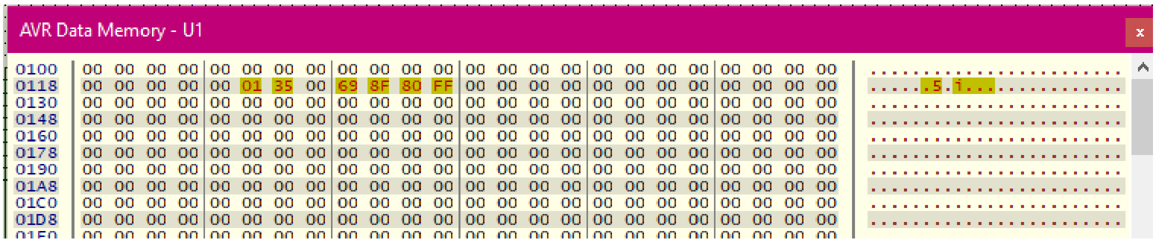
Notice that the last logged result was 0x69 which is equivalent to 0110 1001 and that is exactly what is seen at READ_OUT.



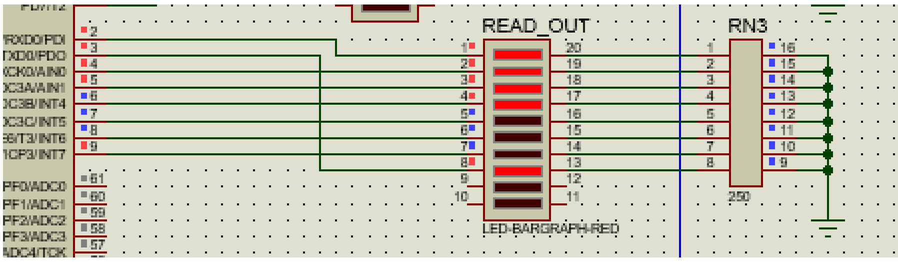
CASE 5 (a):

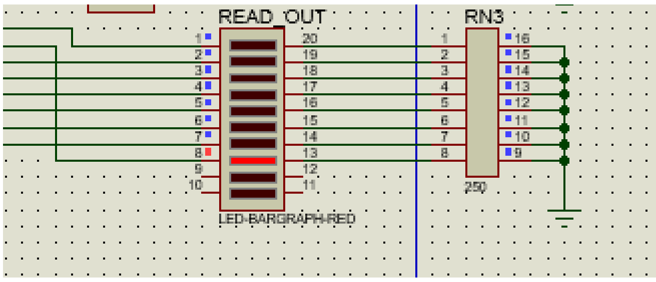
Memory dump asserted.

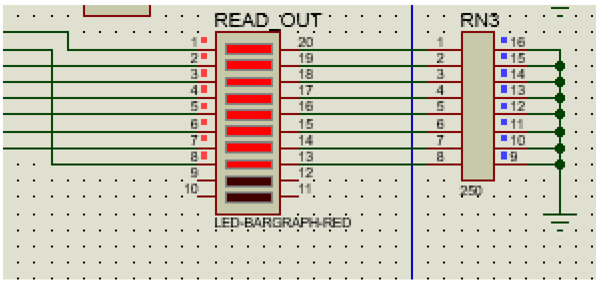
AVR MEMORY looks like:



(note that 0x01, 0x35 and 0x69 are some garbage values that weren't added by us)



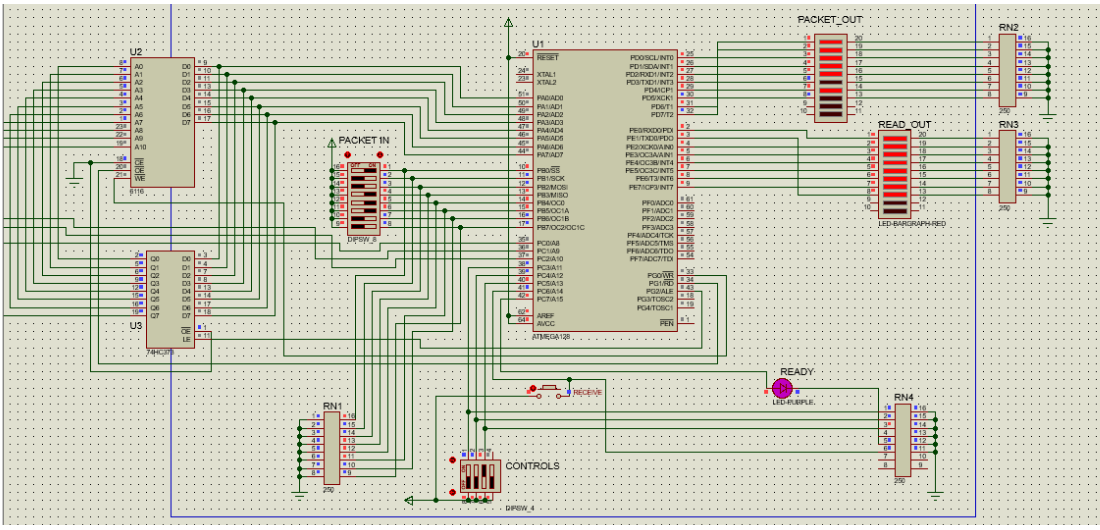




CASE 5 (b):

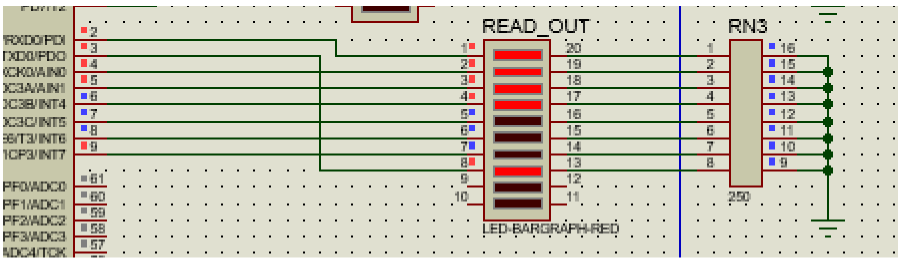
Memory dump and last entry asserted.

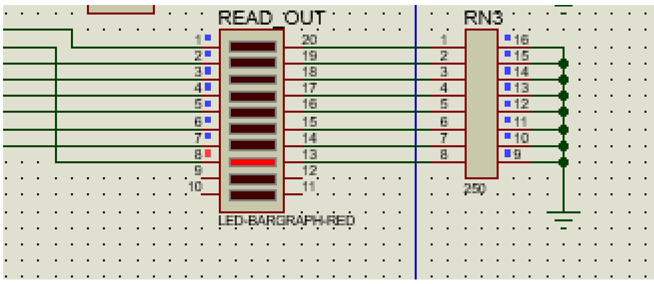
If only last entry is asserted, as expected the output is FF

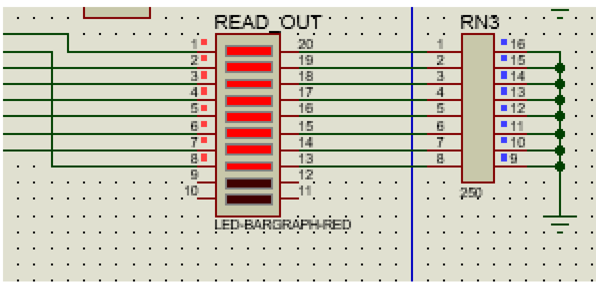


But

if both memory dump and last entry are asserted, the output should be the same as CASE 5 (a). Since last entry is ignored if memory dump is active.







Case 6:

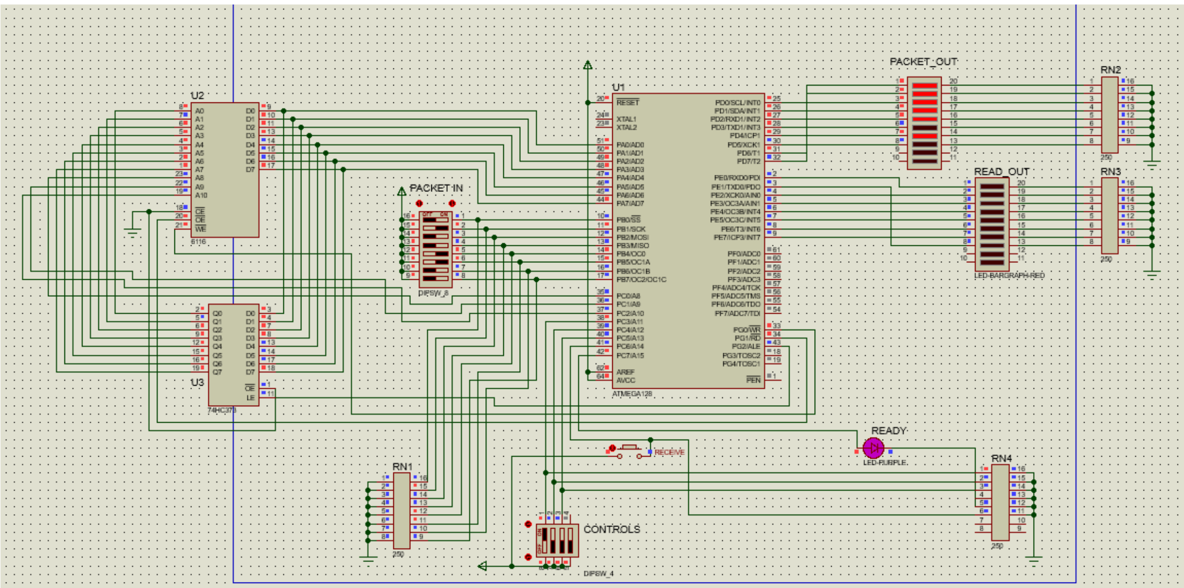
Writing to external memory

For this purpose we added this directive just to check if it would write to external memory:

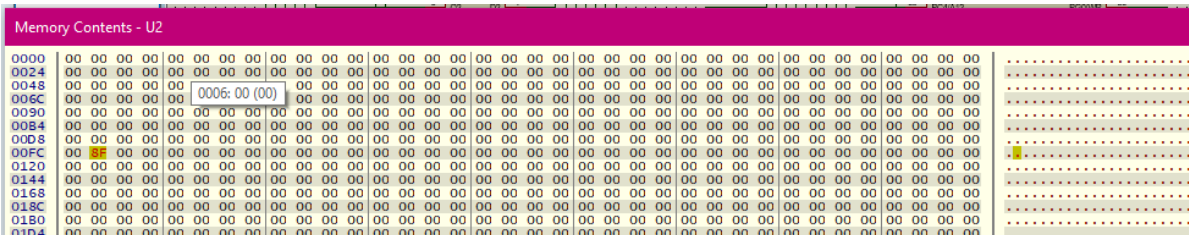
```
.EQU IMEM_START = 0x18FD
```

We sent a data packet of 1000 1111 (8F) followed by correct log request (00110010).

Acknowledge signal was also observed:



As expected, it does write to the external memory and at the correct location:



Conclusion:

We have tested around 10 different cases and can safely say that our code and design is working exactly according to the instructions given in the lab manual. All in all, this was a great lab module that equipped us with the tools and techniques necessary for much more complex system design.