Shane Bolding

Shb7@students.uwf.edu

COP4634: Sys & Net 1

Hungry Lizard Crossing

11-10-19

Objectives:

The object of this lab is to look at the bus cycles of the HC11 and watch as they change between an address bus and a data bus when needed.

Introduction:

To do this lab we will be running a looped program that runs indefinitely and watching the address/data ports using the logic analyzer paired with the logic port software. This should show us a logic state across the timing of the program in the loop.

Procedure:

We need to copy the program from the lab handout and then flash it to the HC11. Then we plug the E-clock, Address Strobe, R/W* Signal, and Address from A15 to AD0 from port C into the Logicport. Once this is done, we then run the HC11 and capture the Logicport reading when it triggers at address $B600, where the program is stored. This should all be done with the HC11 set in extended mode with grounding MODA and MODB. Below is the code that should be put to the board.

```
HPRIO   EQU $103C
MASK    EQU $F5

        ORG $0200
M       RMB 1; RESERVE 1 BYTE FOR M AT BEGINNING OF EXTERNAL RAM
N       RMB 1; RESERVE 1 BYTE FOR N AT NEXT MEMORY LOCATION
SUM     RMB 1; RESERVE 1 BYTE FOR SUM AT NEXT MEMORY LOCATION
        ORG     $B600
        LDAA    #MASK; CHANGE HPRIO FOR INTERNAL READ VISIBILITY
        STAA    HPRIO
        LDAA    #5      ; INITIALIZE M TO 5
        STAA    M
        LDAA    #4      ; INITIALIZE N TO 4
        STAA    N
LOOP
        LDAA M  ; LOAD M INTO ACCA
        LDAB    N       ; LOAB N INTO   ACCB
        ABA     ; ACCA = ACCA + ACCB
        STAA    SUM     ; STORE SUM
        BRA     LOOP    ; LOOP FOREVER
        SWI
```

One should notice the storing of HPRIO. This changes the value of the HPRIO and allows us to see the extended mode in Logicport once this is called at $B604 as shown below.

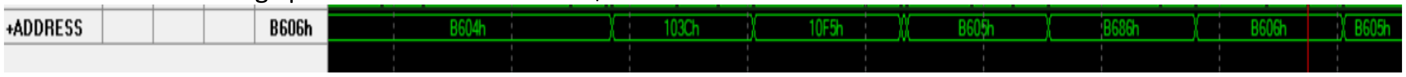


Figure 1 Shows the switch from B604 to the address of the HPRIO $103C

We were also supposed to create a table of the program showing cycle by cycle of the program loop to compare and check the Logicport results with. This is my table below.

	Cycle	Address	Data	R/W*
LDAA	1	$B600	$86	1
	2	$B601	$F5	1
STAA	3	$B602	$B7	1
	4	$B603	$10	1
	5	$B604	$3C	1
	6	$103C	$F5	0
LDAA	7	$B605	$86	1
	8	$B606	$5	1
STAA	9	$B607	$B7	1
	10	$B608	$2	1
	11	$B609	$0	1
	12	$200	$5	0
LDAA	13	$B60A	$86	1
	14	$B60B	$4	1
STAA	15	$B60C	$B7	1
	16	$B60D	$2	1
	17	$B60E	$1	1
	18	$201	$4	0
LDAA	19	$B60F	$B6	1
	20	$B610	$2	1
	21	$B611	$0	1
LDAB	22	$B612	$F6	1
	23	$B613	$2	1
	24	$B614	$1	1
ABA	25	$B615	$1B	1
STAA	26	$B616	$B7	1
	27	$B617	$2	1
	28	$B618	$2	1
	29	$202	$0	0
BRA	30	$B619	$20	1
	31	$B61A	$F4	1

Figure 2 Table of instruction and its Address/Data bus (Question 1)

With this table we will compare it to the results from the Logicport screenshots below.

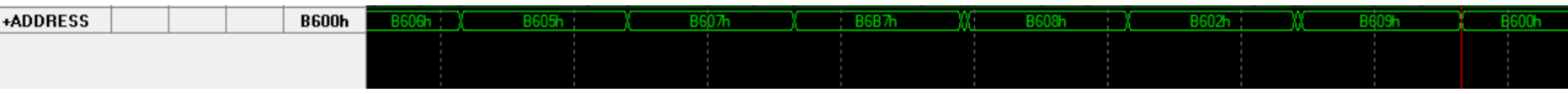


Figure 3 NOTICE: B607:B7; B608:02; B609:00

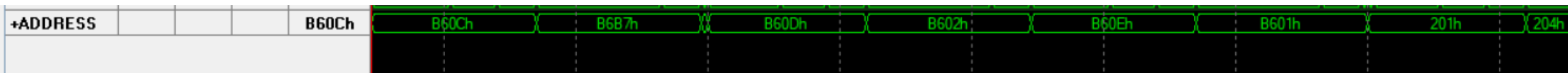


Figure 4 NOTICE: B60C:B7; B60D:02; B60E:01; 201:4

.

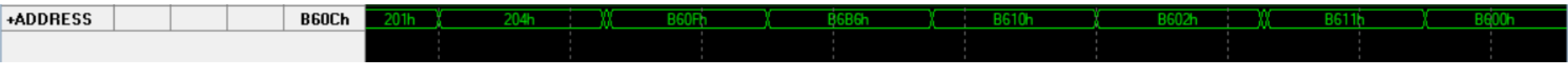
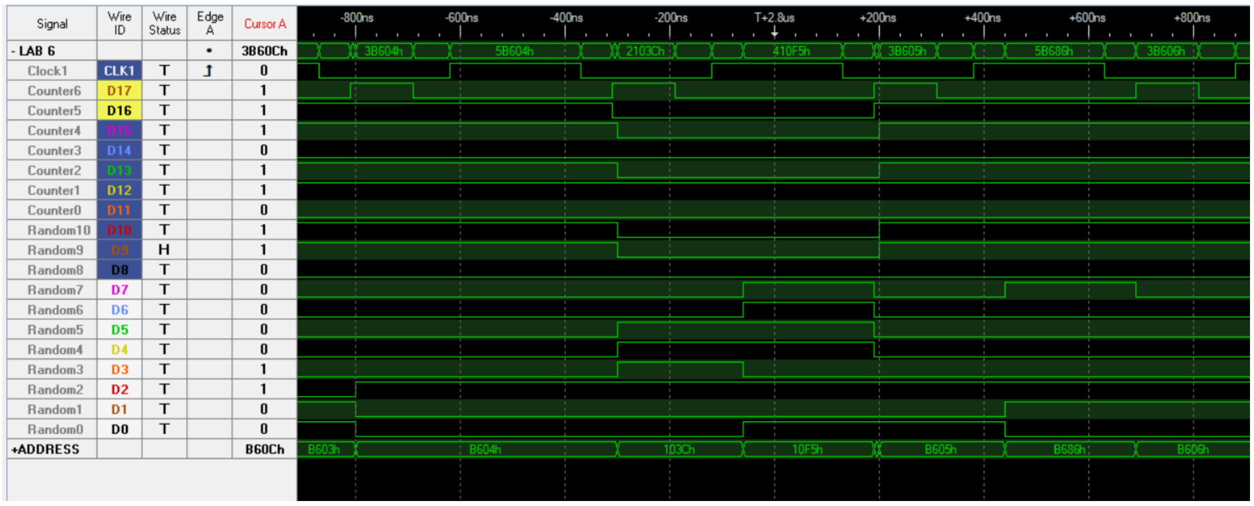


Figure 5 NOTICE: B60F:B6; B610:02; B611:00

These screenshots show that the Address/Data bus changed to their desired values when needed. We also can look at the exact time it changes due to the AS signal and the E-clock signal in the image below.



Signal	Wire ID	Wire Status	Edge A	Cursor A
- LAB 6			•	3B60Ch
Clock1	CLK1	T	↥	0
Counter6	D17	T		1
Counter5	D16	T		1
Counter4	D15	T		1
Counter3	D14	T		0
Counter2	D13	T		1
Counter1	D12	T		1
Counter0	D11	T		0
Random10	D10	T		1
Random9	D9	H		1
Random8	D8	T		0
Random7	D7	T		0
Random6	D6	T		0
Random5	D5	T		0
Random4	D4	T		0
Random3	D3	T		1
Random2	D2	T		1
Random1	D1	T		0
Random0	D0	T		0
+ADDRESS				B60Ch

Figure 6 The AS (Counter6) blips and the Address/Data bus (Random 7 - 0) reads as data now

As you can see these picture shows that the Address bus switched to the data bus during the AS blip showing us that the Logicport is showing a correct representation of the HC11 buses, ergo us achieving the objective of the lab.

Conclusion:

This lab showed us that we can use subroutines to organize our code and declutter the main program. It also showed us that keeping up with the stack pointer, sp, is important to the running of subroutines. This is because if we mess up the sp during a subroutine we may lose our way back to the main program and possible end up in illicit memory.