

# CSE



Department of Computer Science & Engineering

## **AMD 2910® Micro-program Address Sequencer**

### **CSCE 612 – Project #1 Specification**

VLSI Design Lab  
Department of Computer Science and Engineering  
University of South Carolina  
Columbia, SC 29208 USA

Dr. James P. Davis  
jimdavis@cse.sc.edu

Revision: v.05 – March 2003

## 1. Problem Description

The Am2910 is a micro-program address sequencer intended for use in high-speed microprocessor applications. Figure 1 shows a simplified block diagram of Am2910.

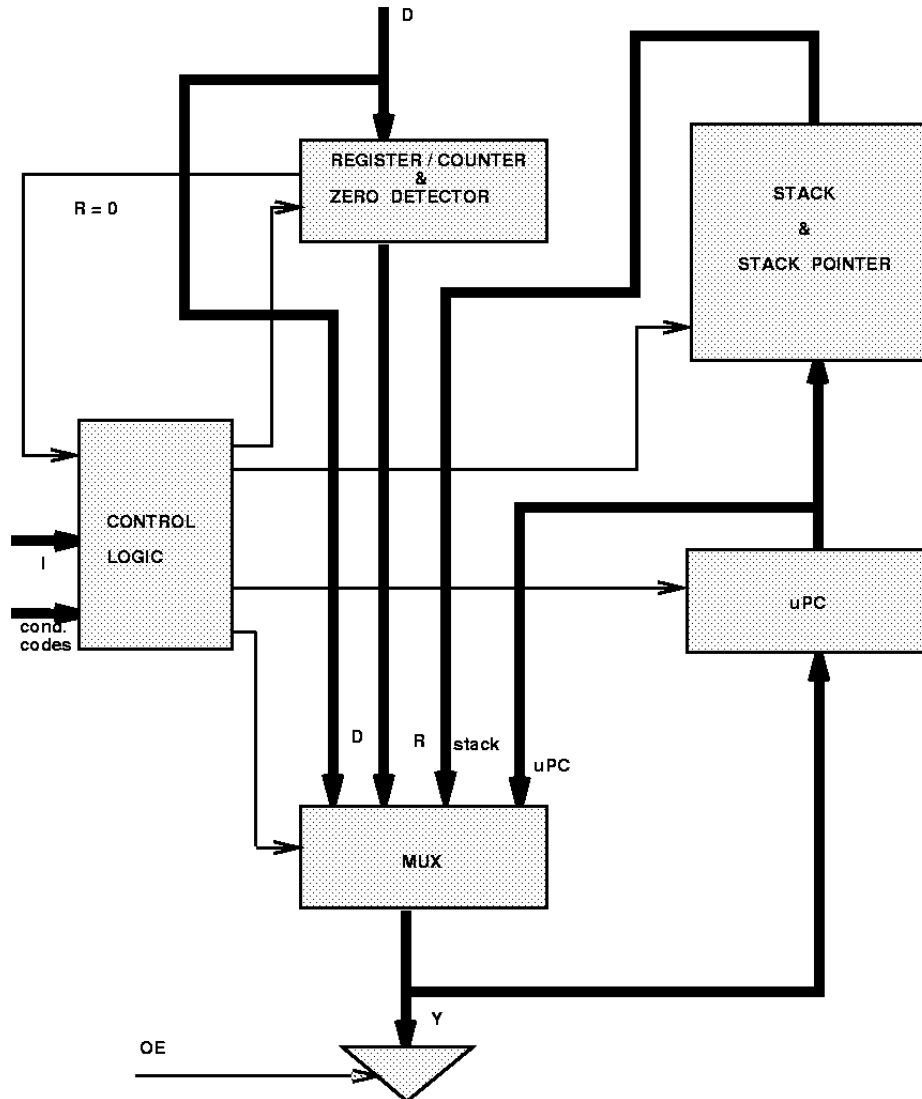


Figure 1. Block diagram of Am2910.

The Am2910 has a four-input multiplexer that is used to select either the register/counter (R), direct data input (D), microprogram counter (uPC) or the top of stack (TOS) as the source of the next microinstruction address.

The microprogram counter, (uPC), is composed of a 12-bit incrementer followed by a 12-bit register. The uPC can be used in either of two methods: When the carry-in, CI, to the incrementer is HIGH, the microprogram register is loaded on the next clock cycle with the current Y output word plus one ( $Y + 1 \rightarrow uPC$ ). Sequential microinstructions are thus executed.

When CI is LOW, the incrementer passes the Y output word unmodified so that uPC is reloaded with the same Y word on the next clock cycle (Y --> uPC). The same microinstruction is thus executed any number of times.

The register/counter is operated during some microinstructions as a 12-bit down counter, with "R = 0" available as a microinstruction branch test criterion. This provides efficient iteration of microinstructions. The register/counter is arranged such that if it is preloaded with a number N and then used as a loop termination counter, the sequence will be executed exactly N + 1 times. Figure 3 shows the REGCNT pin description.

The stack is a 5-word by 12-bit register file. The stack is used to provide return address linkage when executing micro-subroutines or loops. The stack contains a built-in stack pointer (SP) that always points to the last file word written. This allows stack reference operations (looping) to be performed without a pop.

The stack pointer operates as an up/down counter. During some microinstructions, the PUSH operation may occur. This causes the stack pointer to increment and the file to be written with the required return linkage. On the cycle following the PUSH, the return data is at the new location pointed to the stack pointer.

During some microinstructions, a POP operation may occur. The stack pointer decrements at the next rising clock edge following a POP, effectively removing old information from the top of the stack.

The stack pointer linkage is such that any sequence of pushes, pops or stack references can be achieved. At RESET (instruction 0), the depth of nesting becomes zero. For each PUSH, the nesting depth increases by one; for each POP, the depth decreases by one. The depth can grow to five. After a depth of five is reached, FULL\_BAR goes LOW. Any further PUSHes onto a full stack overwrite information at the top of the stack, but leave the stack pointer unchanged. This operation will usually destroy useful information and is normally avoided. A POP from an empty stack may place non-meaningful data on the Y outputs, but is otherwise safe. The stack pointer remains at zero whenever a POP is attempted from a stack already empty.

The device provides three-state Y outputs. These can be particularly useful in designs requiring automatic checkout of the processor. The micro-program sequencer outputs can be forced into high-impedance state, and pre-programmed sequences of microinstructions can be executed via external access to the address lines.

## 1.1 Pin description

Name	Type	Width	Description
I	I	4	Four-bit signal for 16 instructions
CCEN	I	1	Conditional Code Enable bit
CC	I	1	Conditional Code Bit
RLD	I	1	Unconditional load bit for register/counter
CIN	I	1	Carry-in bit for microprogram counter
OE	I	1	Tri-state driver control bit
Clk	I	1	Clock

D	I	12	12-bit data input
Y	O	12	12-bit data output
PL BAR	O	1	Pipeline register output enable
VECT BAR	O	1	Vector output enable
MAP BAR	O	1	Mapping PROM output enable
FULL	O	1	Stack full flag

## 1.2 Instruction Summary

The Am2910 has a 4-bit instruction. The actions performed in various modules are controlled by the instruction as well as by the two *condition code* bits (CCEN\_BAR and CC\_BAR) and by the contents of the register/counter. The following table lists the actions performed by each module.

I		Name of instructions	Register Content	FALL		PASS		Register Counter	Enable Signal
				Y	STACK	Y	STACK		
0	JZ	Jump to zero (reset)	X	0	Clear	0	Clear	Hold	PL
1	CJS	Conditional jump to subroutine via PL	X	uPC	Hold	D	Push	Hold	PL
2	JMAP	Jump via map ( address supplied by mapping proms)	X	D	Hold	D	Hold	Hold	MAP
3	CJP	Conditional jump via PL (pipeline register address)	X	uPC	Hold	D	Hold	Hold	PL
4	PUSH	Push stack / conditional load register-counter	X	uPC	Push	uPC	Push	**	PL
5	JSRP	Conditional jump to subroutine via reg-cntr or PL address	X	R	Push	D	Push	Hold	PL
6	CJV	Conditional jump via Vect. (vector address)	X	uPC	Hold	D	Hold	Hold	VECT
7	JRP	Conditional jump via reg-cntr or PL address	X	R	Hold	D	Hold	Hold	PL

I		Name of instructions	Register Content	FALL		PASS		Register Counter	Enable Signal
				Y	STACK	Y	STACK		
8	RFCT	Repeat loop if reg-cntr != 0 (address from top of stack )	!= 0	TOS	Hold	TOS	Hold	DEC	PL
			!=	UPC	POP	uPC	POP	Hold	PL
9	RPCT	Repeat loop if reg-cntr != 0 ( address from PL)	!= 0	D	Hold	D	Hold	DEC	PL
			= 0	uPC	Hold	uPC	Hold	Hold	PL
10	CRTN	Conditional return from subroutine	X	uPC	Hold	TOS	POP	Hold	PL
11	CJPP	Conditional jump via PL (pipeline register address)	X	uPC	Hold	D	POP	Hold	PL
12	LDCT	Load reg-cntr and	X	uPC	Hold	uPC	Hold	Load	PL

		continue							
13	LOP	Test end of loop	X	TOS	Hold	uPC	POP	Hold	PL
14	CONT	Continue	X	uPC	Hold	uPC	POP	Hold	PL
15	TWB	Three way branch	!= 0	TOS	Hold	uPC	POP	DEC	PL
			= 0	D	POP	uPC	POP	Hold	PL

**NOTE:**

**\*\* If FAIL then HOLD register counter else LOAD it.**

- “TOS” = Top of Stack
- “Fail” == CCEN is 0 and CC is 1
- “Pass” = CCEN is 1 and CC is 0
- “R” = Register Counter
- “uPC” = Microprogram Counter
- “!= ” = Not Equal to
- “I” = 4-bit Instruction Input
- “X” = Don’t Care

**2 Design Partitioning**

The detailed partitioned block diagram (created using blockHDL™) for the am2910 is shown in Figure 2.

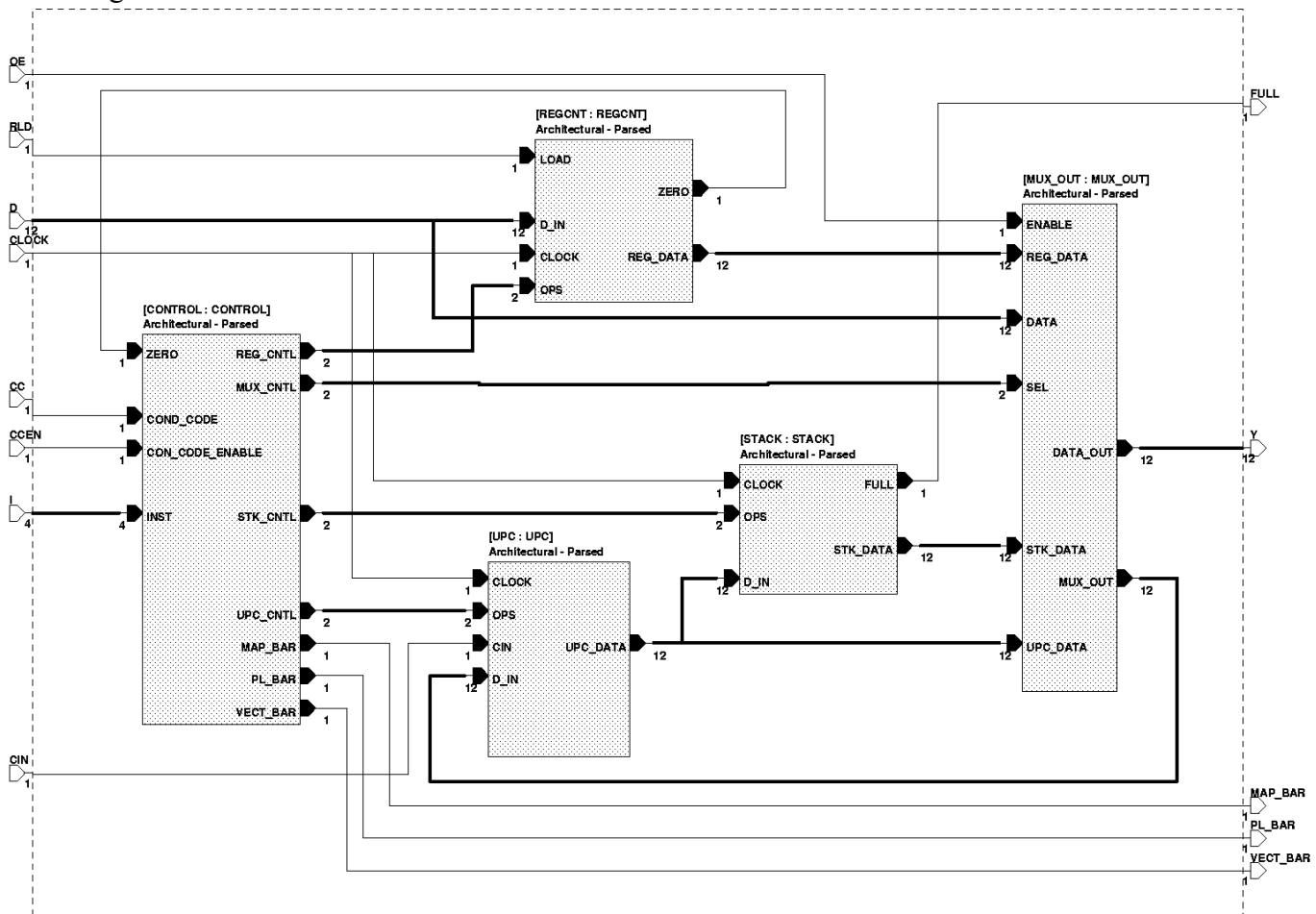
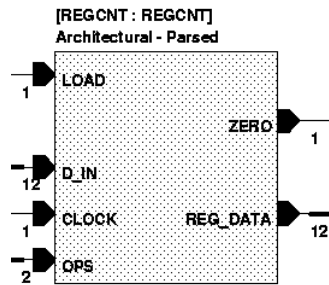


Figure 2. System-level block diagram.

## 2.1 Register and counter block ( REGCNT)

The register/counter consists of 12 D-type flip-flops with a common clock. When its load control, LOAD is LOW, new data is loaded on a positive clock edge. A few instructions include load or decrement. The output of the register/counter, (REG\_DATA), is available to the Multiplexer as a source for the next microinstruction address. The direct input (D\_IN) furnishes a source of data for loading the register/counter.



Name	Width	Type	Description
LOAD	1	I	Enable signal for load data into the register
D_IN	12	I	Data bus to the register
CLOCK	1	I	Clock signal
OPS	2	I	Control signal for the register (Hold, Load, and Decrement).
ZERO	1	O	Zero flag is high when the register is zero.
REG_DATA	12	O	Output of the register

Figure 3. Pin description of the REGCNT.

## 2.2 Stack

The Stack has four operations. These are Hold, Clear, Pop, and Push. The stack pointer operates as an up/down counter. During some micro- instructions, the PUSH operation may occur. This causes the stack pointer to increment and the file to be written with the required return linkage. On the cycle following the PUSH, the Stack Pointer points to the return data at the new location pointed to by the stack pointer.

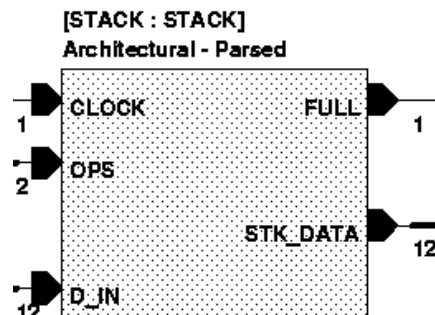


Figure 4. Pin description of the Stack.

Name	Width	Type	Description
CLOCK	1	I	Clock signal
OPS	2	I	Control signal for the Stack ( Hold, Clear, Pop, and Push )
FULL	1	O	Stack full signal
STK_DATA	12	O	Data bus from the Stack
D_IN	12	I	Data bus to Stack

### 2.3 Microprogram Counter (uPC)

The microprogram counter, (uPC), is composed of a 12-bit incrementer followed by a 12-bit register. The uPC can be used in either of two ways : When the carry-in, CI, to the incrementer is HIGH, the microprogram register is loaded on the next clock cycle with the current Y output word plus one ( $Y + 1 \rightarrow \text{uPC}$ ). Sequential microinstructions are thus executed. When CI is LOW, the incrementer passes the Y output word unmodified so that uPC is reloaded with the same Y word on the next clock cycle ( $Y \rightarrow \text{uPC}$ ). The same microinstruction is thus executed any number of times.

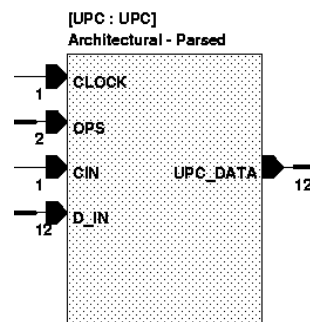


Figure 5. Pin description of the micro-program counter

Name	Width	Type	Description
CLOCK	1	I	Clock signal
OPS	2	I	Control signal for the Stack (Clear and Count )
CIN	1	I	Increment signal
D_IN	12	I	Data bus to the uPC
UPC_DATA	12	O	Data bus output

## 2.4 Output Multiplexer

The four-input Multiplexer is used to select among the following: the register/counter (R), direct data input (D), micro-program counter (uPC) or the top of stack (TOS) as the source of the next microinstruction address.

Name	Width	Type	Description
ENABLE	1	I	Tri-state output enable
REG_DATA	12	I	Data bus from the register/counter
DATA	12	I	Data bus from outside the chip.
SEL	2	I	Output selector (Data, RegCnt, uPC, and Stack )
STK_DATA	12	I	Data bus from the Stack
UPC_DATA	12	I	Data bus from the microprogram counter
DATA_OUT	12	O	Data output from the chip
MUX_OUT	12	O	Data bus for loading uPC with new data.

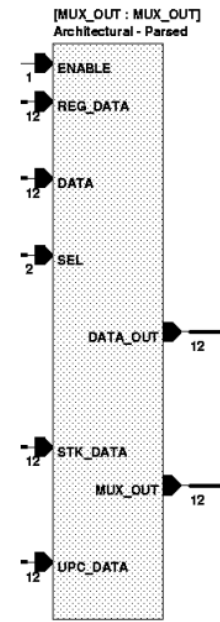
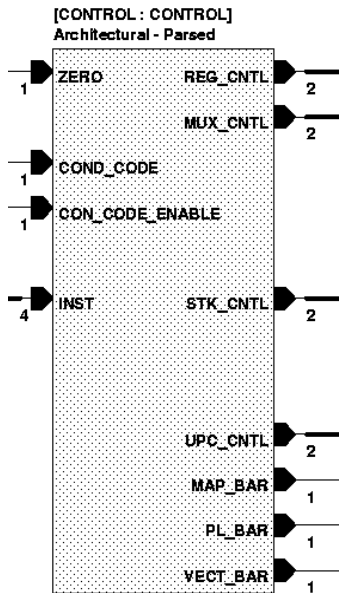


Figure 6. Pin description of the output Multiplexer

## 2.5 Main Controller

The main controller decodes the instruction and sends control signals to the stack, micro-program counter, register/counter, and output Multiplexer. For each instruction, one and only one of the three outputs PL\_BAR, MAP\_BAR, VECT\_BAR is LOW



Name	Width	Type	Description
ZERO	1	I	Zero flag indicates that register is zero
CONC_CODE	1	I	Conditional Code Bit
CONC_CODE_ENABLE	1	I	Conditional Code Enable bit
INST	4	I	Four Bit signal for Instructions
RED_CNTRL	2	O	Control signal for register (Hold, Load and Decrement)
MUX_CNTRL	2	O	Output selector (Data, RegCnt, uPC and Stack)
STK_CNTRL	2	O	Control signal for the Stack ( Hold, Clear, Pop and Push)
UPC_CNTRL	2	O	Control signal for the counter (Clear and Count)
MAP_BAR	1	O	Mapping PROM output enable
PL_BAR	1	O	Stack full flag
VECT_BAR	1	O	Vector output enable

Figure 7. Pin description of the main controller

### 3. Project Schedule

Figure 8 shows the effort distribution of the project. You should use this as a guide when planning your work on the project.

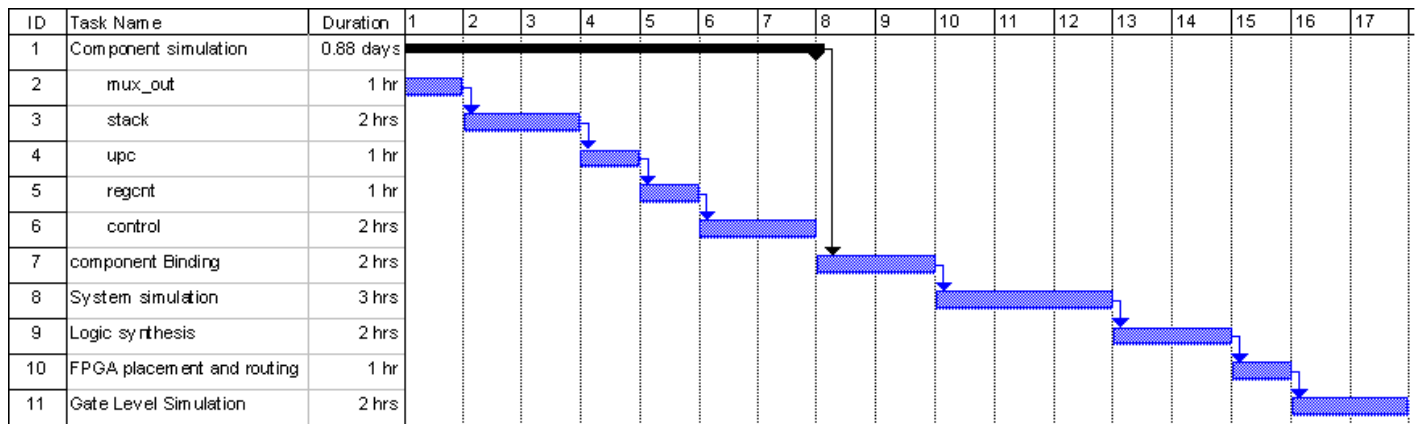


Figure 8. Project Work Breakdown Structure (WBS).

This WBS plan has been assessed across a number of different executions of this project in this and similar courses here at USC. Note that we will use this as a guide, to show the division of labor, and how much time you should spend on each effort. We will attempt to support the logic synthesis and the FPGA layout tasks using the Synopsys® FPGA Compiler II® and Xilinx® ISE tool sets; however, if we are not able to make the tools available, or if time gets too compressed, we will not do these finals tasks for the project. I will be interested to see how our class does against this timetable, as it was the subject of research into design productivity (which Dr. Kobayashi and I, along with some graduate students, are continuing to investigate).