EPS448 SAM EPLD

Stand-Alone Microsequencer

September 1991, ver. 3

Data Sheet

Features

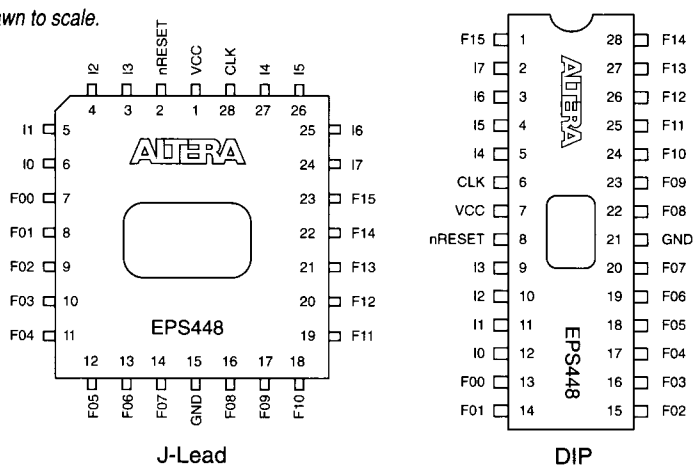
- User-configurable Stand-Alone Microsequencer (SAM) for implementing high-performance controllers
- On-chip reprogrammable microcode EPROM up to 448 words deep
- 15 × 8-bit stack
- Loop counter
- Prioritized multiway control branching
- 8 general-purpose branch-control inputs and 16 general-purpose control outputs
- Cascadable to expand the number of outputs or states
- Low-power CMOS technology
- Available in 28-pin, 300-mil windowed ceramic dual in-line packages (CerDIP) and 28-pin windowed ceramic and one-time-programmable (OTP) plastic J-lead chip carriers (JLCC and PLCC)
- Clock frequencies up to 30 MHz
- High-level support with SAM+PLUS design tools includes Altera State Machine Input Language (ASMILE), Assembly Language (ASM), SAM Design Processor (SDP), and SAMSIM functional simulator.

General Description

The EPS448 EPLD is a function-specific, user-configurable Stand-Alone Microsequencer (SAM). It is available in a 28-pin windowed ceramic and OTP plastic J-lead chip carrier, and 300-mil windowed ceramic DIP packages. See Figure 1.

Figure 1. EPS448 Package Pin-Out Diagrams

Package outlines not drawn to scale.



The on-chip EPROM of each EPS448 device (up to 448 words) is integrated with branch-control logic, a pipeline register, a stack, and a loop counter. This generic microcoded architecture can efficiently implement a broad range of high-performance controllers, from state machines to waveform-generation applications.

The 1.2-micron CMOS EPROM technology allows the EPS448 EPLD to operate at 30-MHz clock frequency while still benefitting from low CMOS power consumption. This technology also facilitates 100% generic testability, which eliminates the need for post-programming testing.

Altera's SAM+PLUS software provides design entry, logic optimization, and functional simulation for EPS448 designs. With SAM+PLUS, designs are entered in either state machine or microcoded format. The software automatically performs logic minimization and design fitting. The designer can then simulate the design or program it directly to create customized working silicon. Programming takes only a few minutes with standard Altera programming hardware and LogicMap II software. New users can purchase the complete PLDS-SAM Development System with programming hardware included; PLS-SAM is a software-only package for existing Altera systems.

Applications

Ideal EPS448 applications include programmable sequence generators (i.e., state machines), bus and memory control functions, graphics and DSP algorithm controllers, and other high-performance control logic. EPS448 devices can be cascaded horizontally for greater output capabilities and vertically for deeper microcode memory. See *Application Brief 65 (Vertical Cascading of EPS448 SAM EPLDs).*

EPS448 as a State Machine

EPS448 architecture easily implements synchronous state machines. The device's internal EPROM memory and pipeline register allow up to 448 unique states to be specified. Its branch-control logic allows single-clock, multiway branching based on the eight inputs, the current device state, and the user-defined transition conditions. Design entry is simplified with the Altera State Machine Input Language (ASMILE) supported by SAM+PLUS software. This high-level language uses `IF-THEN` statements to define state transitions and truth tables to define or tri-state the outputs on a state-by-state basis.

EPS448 as a Microcoded Controller

EPS448 architecture provides several advanced features that make it suitable for use as a complex microcoded controller. The EPS448 EPLD's 448-word on-chip EPROM is integrated with a microcode sequencer consisting of branch-control logic, a stack, and a loop counter. The branch-control logic—

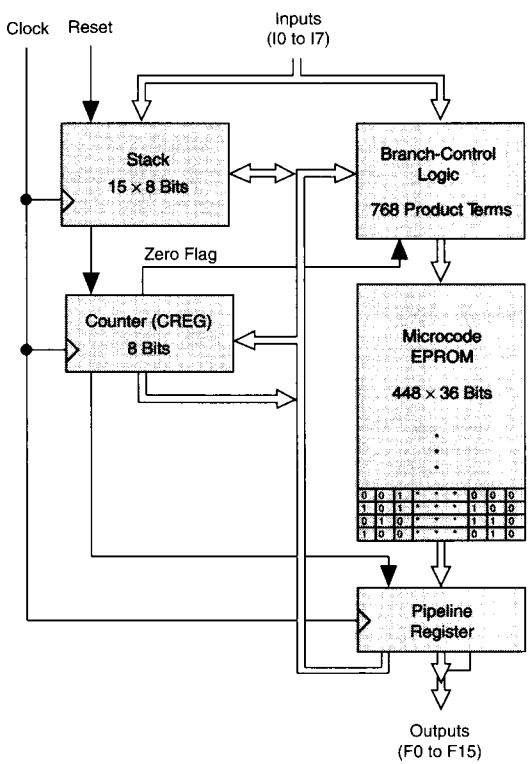
fed by the 8 general-purpose inputs, the counter, the stack, and the pipeline register—provides flexible, multiway microcode branch capability in a single clock cycle, enhancing throughput beyond that of conventional controllers or sequencers.

For microcoded controllers, SAM+PLUS software offers the high-level Assembly Language (ASM) design entry format. This language consists of powerful instructions (i.e., opcodes) that easily implement conditional branches, subroutine calls, multi-level `FOR-NEXT` loops, and dispatch functions (i.e., branching to an externally specified address). For more information, see "Instruction Set" later in this data sheet.

Functional Description

As shown in Figure 2, the EPS448 EPLD consists of microcode EPROM, a 36-bit pipeline register, branch-control logic, a 15 × 8-bit stack, and an 8-bit loop counter.

Figure 2. EPS448 Block Diagram



The branch-control logic generates the address of the next state and applies it to the microcode memory. The outputs of the microcode memory represent user-defined outputs and internal control values associated with the next state. These new values are clocked into the pipeline register on the leading edge of the clock and become the current state. The new values in the pipeline register—along with the counter, stack, and inputs—are used by the branch-control logic to generate the new next-state address.

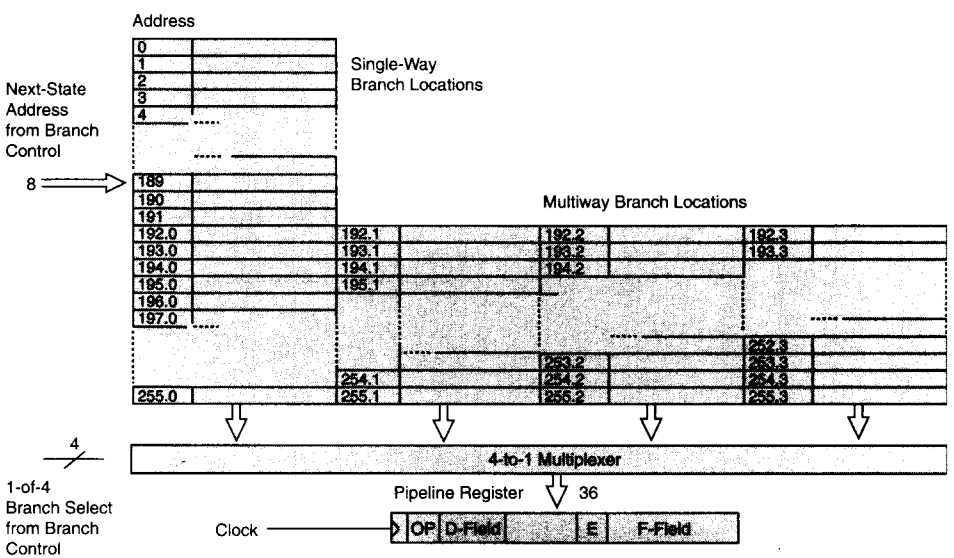
Microcode EPROM and Pipeline Register

The microcode EPROM is organized into 448 36-bit words, each of which can be viewed as a single state location. Each of the 36 bits is divided into the following categories:

F-field (16 bits) consists of user-defined outputs at device pins.
Q-field (8 bits) provides the next-state address.
D-field (8 bits) is a general-purpose field used either as a constant or as an alternative next-state address.
OP-field (3 bits) contains the instruction (opcode).
E-field (1 bit) enables or tri-states the device outputs.

As shown in Figure 3, the microcode memory is organized as 256 addresses. Addresses 0 through 191 contain a single 36-bit word, which is associated with the desired next state. This state information is clocked into the pipeline register on the rising edge of the clock, and the outputs become valid one clock-to-output delay (t_{CO}) later.

Figure 3. Microcode Memory



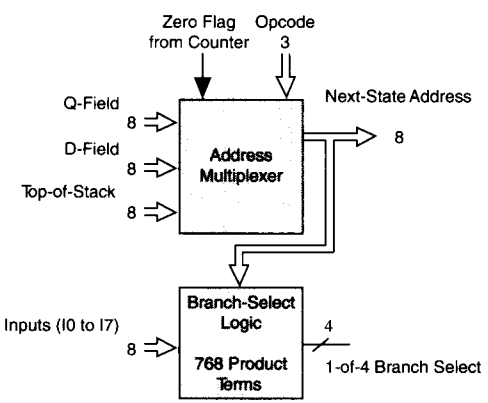
Addresses 192 through 255 access 4 unique 36-bit words, each of which corresponds to a different possible next state. (The extensions .0, .1, .2, and .3 are added to the addresses to distinguish the four states.) These 64 addresses make up the multiway branch locations, and are used to perform single-clock, four-way branching. Whenever the next-state address falls within the multiway branch locations, the branch-control logic makes the necessary 1-of-4 selection based on the next-state address and user-defined input conditions.

Branch-Control Logic Block

The branch-control logic is the key to the high-performance sequencing ability of the EPS448 EPLD. This block determines the next state to be clocked into the pipeline register, based on the current status of the pipeline register, the counter, the stack, and the eight input pins.

The branch-control logic is divided into two segments: the address multiplexer and the branch-select logic. See Figure 4.

Figure 4. Branch-Control Logic



The address multiplexer provides the next-state address to the microcode memory. The next-state address can come from the Q-field, the D-field, or the top-of-stack. The selection is based on the instruction in the pipeline register and the condition of the zero flag from the counter.

The branch-select logic is a programmable logic block with 768 product terms, 16 inputs, and 4 outputs. It is used to perform a 2-, 3-, or 4-way branch based on user-defined input conditions. When the next-state address falls within the multiway branch range of memory—i.e., any address greater than 191—the branch-select logic performs the necessary 1-of-4 selection. When the next-state address is less than 192, no selection is required and the branch-select logic is turned off.

The conditions controlling the multiway branch are defined by the user with a simple IF-THEN-ELSE format, as shown in the following example:

```
IF      (cond3)    THEN   select 201.3
ELSEIF  (cond2)    THEN   select 201.2
ELSEIF  (cond1)    THEN   select 201.1
ELSE                      select 201.0
```

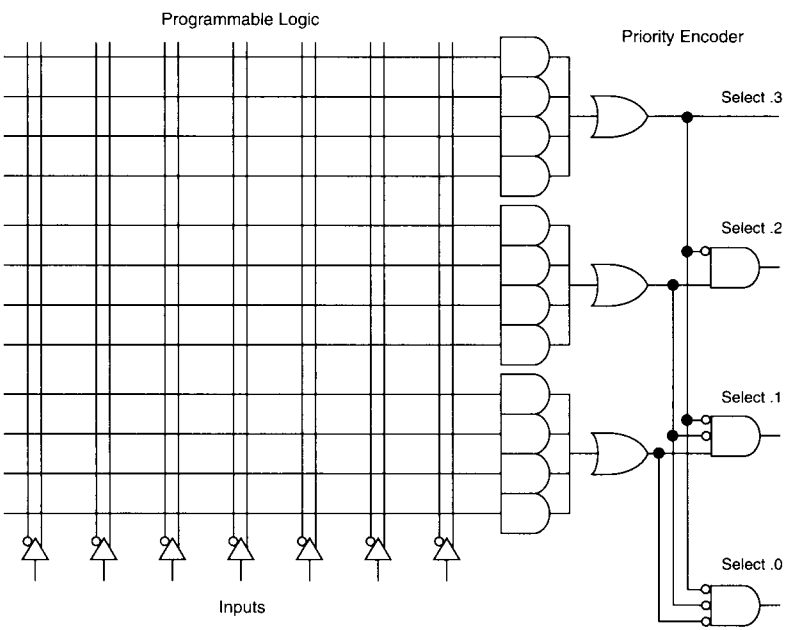
The conditions are prioritized so that if the first condition (i.e., cond3) is met, then microword 201.3 is selected and clocked into the pipeline register, regardless of the results of cond2 and cond1. If none of the conditions are met, then microword 201.0 is clocked into the pipeline register.

The three conditional expressions are user-defined. They may contain any logical equation that is based on the inputs and can be reduced to four product terms, as shown in the following example:

```
I1 * /I2 * /I4
+ I3 * /I4 * /I5 * /I6 * /I7
+ I0
+ I2 * /I4 * /I5
```

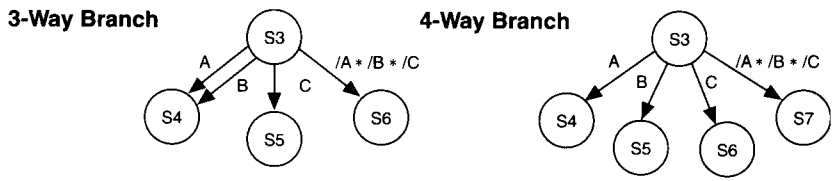
A unique set of 12 product terms is present in each of the 64 available multiway branch locations for a total of 768 product terms. See Figure 5.

Figure 5. Branch Logic in a Multiway Branch Location



The EPS448 EPLD is designed so that the number of available product terms is always sufficient for a design. Prioritization provides an effective product-term count of more than 12 per location. A tradeoff between the number of product terms and the number of possible branches can be made simply by placing identical state information in 2 locations, as shown in Figure 6.

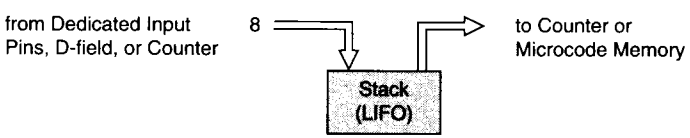
Figure 6. Multiway Branching vs. Product-Term Needs



Stack

The EPS448 stack is a Last-In First-Out (LIFO) arrangement that consists of 15 8-bit words. The top of stack may be used as the next-state address or popped into the counter. Values may be pushed onto the stack from either the D-field in the pipeline register or from the counter. Thus subroutines, nested loops, and other iterative structures may be implemented efficiently. The logic levels on the 8 dedicated input pins may also be pushed onto the stack to allow external address specification in a dispatch function or to externally load the counter. See Figure 7.

Figure 7. Stack



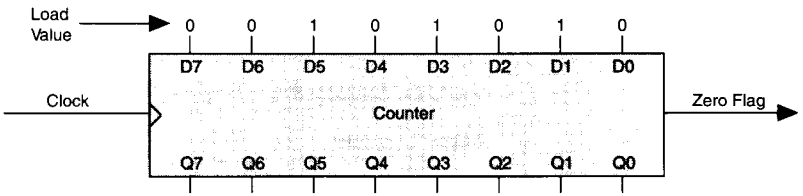
The pushing or popping of the stack occurs on the leading edge of the clock. The stack is "zero-filled" so that a pop from an empty stack will reset all 8 bits to zero. On the other hand, a push to an already full stack will write over the top-of-stack, leaving the other 14 values unchanged.

Loop Counter

The EPS448 EPLD contains an eight-bit loop counter called count register (CREG), which is useful for controlling timing loops and determining branch-control functions. The CREG is a down counter that may be loaded directly from the D-field of the pipeline register or from the top-of-stack.

The value of the CREG may be saved and restored by pushing and popping it to and from the stack. See Figure 8.

Figure 8. Loop Counter (CREG)



The CREG is loaded or decremented on the leading edge of the clock. It will not decrement once it reaches zero, thereby preventing roll-over. A zero flag indicates when the counter has reached zero. This flag is used with the LOOPNZ command to control program flow. (See "Instruction Set" later in this data sheet.) Single-instruction delay loops are easily constructed, and nested loops or delays of arbitrary length may be generated in combination with the stack.

Output Enable Control

Each microcode word contains an OE bit (i.e., the E-field) that enables the outputs when E = 1, and causes high impedance when E = 0. This bit is accessible through instruction set commands provided with SAM+PLUS software. This output-enable capability allows EPS448 EPLDs to be vertically cascaded to increase the number of states.

nRESET Pin

The nRESET pin acts as a master reset for the EPS448 EPLD, causing it to empty the stack, clear the counter, and load the microword at address 0 into the pipeline register. The nRESET signal is useful for system reset or for synchronizing several EPS448 devices that are cascaded vertically or horizontally.

The nRESET signal must be held low for at least three rising clock edges to reset the EPS448 EPLD. An nRESET of one rising clock edge causes the EPS448 device to enter into a supervisor mode; an nRESET of two clock edges leads to an undefined state.

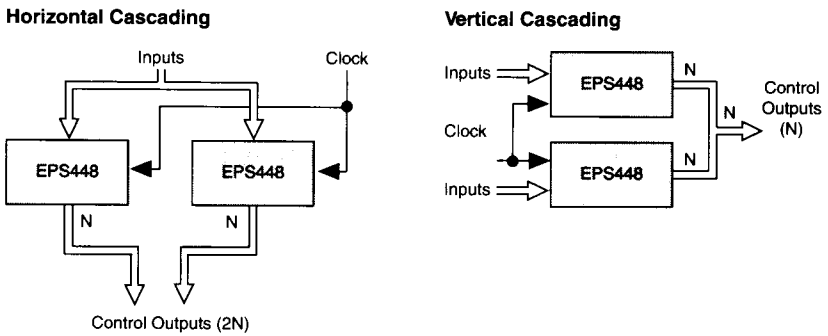
The outputs of the startup address (00 Hex) appear at the pins when the fourth clock edge after nRESET goes low, and are maintained until the third clock edge after nRESET returns to high.

When the EPS448 EPLD is operating in noisy environments, a glitch on the nRESET pin during one setup cycle (t_{SUR}) before the clock edge might initiate a supervisor mode. To prevent this effect, a capacitor of at least 0.1 µF should be connected from the nRESET input to ground.

Horizontal and Vertical Cascading

EPS448 EPLDs, like memory- and bit-slice devices, can be cascaded to provide greater functionality (Figure 9). If an application requires more output lines, two or more EPS448 devices can be cascaded horizontally. Likewise, if an application requires more states, two or more EPS448 EPLDs can be cascaded vertically. In either case, no speed penalty is incurred. The designer can also simultaneously cascade EPS448 devices horizontally and vertically. Designs with horizontal cascading are fully supported by the SAM+PLUS development software. However, vertical cascading requires the designer to make certain tradeoffs to split the design. Refer to *Application Brief 65 (Vertical Cascading of EPS448 SAM EPLDs)* for more information.

Figure 9. Horizontal and Vertical Cascading



Instruction Set

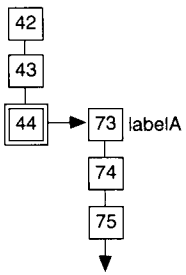
The instruction set used to enter designs for the EPS448 EPLD consists of a compact assortment of powerful commands that allows efficient implementation of multiway branching, subroutines, nested `FOR-NEXT` loops, and dispatch functions. These instructions are used only with Assembly Language (ASM) design entry.

Each command in the instruction set is described and illustrated here. In the following descriptions, `labelA` and `labelB` represent arbitrary labels located in the ASM file. These symbolic labels are converted by the SAM+PLUS software into 8-bit absolute addresses. (SAM+PLUS allows the designer to use the high-level Assembly Language without worrying about the actual values that are placed in the various fields.) The parameter constant is any 8-bit number (0 to 255 decimal, 0 to FF hexadecimal) that represents an address, a mask, or a constant.

For simplicity, it is assumed that the sample destination labels in the following descriptions are not in the multiway branch block. See "Multiway Branching" later in this data sheet for more details about this capability.

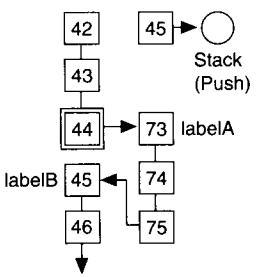
CONTINUE

This command causes execution to continue with the next sequential instruction in the ASM file. In this example, the current address is 44, and CONTINUE instructs SAM+PLUS to go to address 45 in the ASM file.



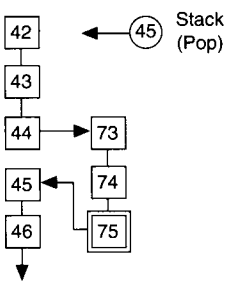
JUMP labelA

This instruction causes execution to branch to the indicated location. In this example, address 44 contains the instruction JUMP labelA ; labelA is located at address 73. The next instruction will come from labelA.



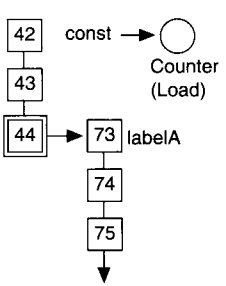
CALL labelA RETURNTO labelB

This instruction pushes the address of labelB onto the stack and makes labelA the next-state address. CALL labelA without the RETURNTO command makes labelB default to the next instruction in the ASM file. In this example, the address location 44 contains the instruction CALL labelA; labelA is located at address 73. The instruction pushes the address of the next instruction (45) onto the stack and causes the next instruction to come from address 73. The RETURN instruction at address 75 returns the execution to address 45. The CALL command is typically used to call a subroutine.



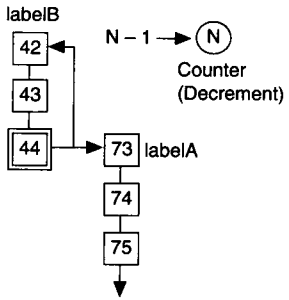
RETURN

This command causes the address of the next instruction to come from the top-of-stack and pops that value off the stack. In this example, the instruction at address 44 calls the subroutine at address 73 and pushes the value 45 onto the stack. The RETURN instruction at address 75 pops the value 45 off of the top-of-stack and causes execution to continue with address 45. RETURN is most frequently used to return from a subroutine.



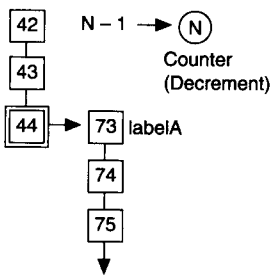
LOADC constant GOTO labelA

This command loads the counter with the specified value and then executes the instruction at labelA. If GOTO is not included in the instruction, labelA defaults to the next instruction in the ASM file. In this example, the instruction LOADC 173D GOTO labelA is located at address 44. This means that the decimal value 173 is loaded into the counter and the next state comes from labelA at address 73. LOADC is typically used to load the counter before entering a FOR-NEXT loop or a wait-state generator.



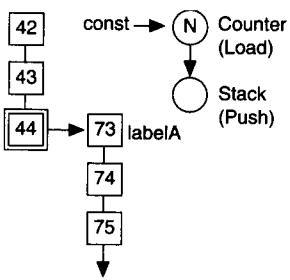
LOOPNZ labelB ONZERO labelA

This instruction jumps to one of two addresses based on the value of the zero flag, and decrements the counter if it is not already zero. If it is zero (i.e., zero flag = 1), the next instruction comes from `labelA`. If it is not zero (i.e., zero flag = 0), the next instruction comes from `labelB`. If the `ONZERO` instruction is not included, `labelA` defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is `LOOPNZ labelB ONZERO labelA`, where `labelB` is located at address 42 and `labelA` at address 73. If the counter is not at zero, the instruction at address 42 is executed and the counter is decremented. If the counter is already at zero, the instruction at address 73 is executed and the counter remains at zero. `LOOPNZ` is typically used to implement `FOR-NEXT` loops.



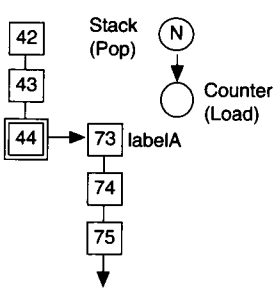
DECNZ GOTO labelA

This command decrements the counter if it is not zero and then jumps to the instruction specified at `labelA`. If `GOTO` is not included in the instruction, `labelA` defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is `DECNZ GOTO labelA`, where `labelA` is located at address 73. The counter is decremented if it is not zero and the next instruction comes from address 73. `DECNZ` is typically used to conditionally decrement the counter.



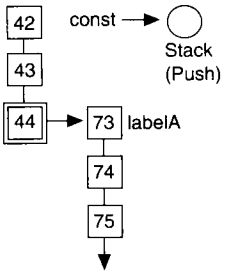
PUSHLOADC constant GOTO labelA

This instruction pushes the current value of the counter onto the stack, loads a new value into the counter, and jumps to `labelA`. If the `GOTO` instruction is not included, `labelA` defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is `PUSHLOADC 153D GOTO labelA`, where `labelA` is located at address 73. The value in the counter is pushed onto the stack, the decimal value 153 is loaded into the counter, and the next instruction comes from address 73. `PUSHLOADC` is useful for implementing `FOR-NEXT` loops.



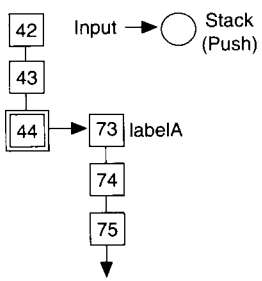
POPC GOTO labelA

This command pops the top-of-stack into the counter and jumps to `labelA`. If the `GOTO` instruction is not included, `labelA` defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is `POPC GOTO labelA`, where `labelA` is located at address 73. The current value at the top-of-stack is removed from the stack (i.e., popped) and loaded into the counter. The next instruction comes from address 73. `POPC` is typically used with the `PUSHLOADC` instruction to implement nested `FOR-NEXT` loops.



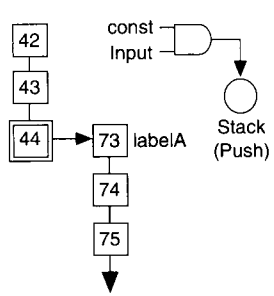
PUSH constant GOTO labelA

This command pushes the value of the constant onto the stack and jumps to labelA. If the GOTO instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is PUSH 34D GOTO labelA, where labelA is located at address 73. The decimal value 34 is pushed onto the stack and the next instruction comes from address 73. PUSH is typically used to store a value on the stack.



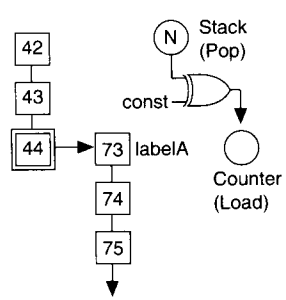
PUSHI GOTO labelA

This instruction pushes the eight inputs (I7 to I0) onto the stack. If the GOTO instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is PUSHI GOTO labelA, where labelA is located at address 73. At the leading edge of the clock, the eight inputs are pushed onto the stack. Typically, address 73 would have a RETURN instruction that would cause execution to jump to the address represented by the recently pushed input pins, implementing a dispatch function. This instruction can also be used to load the counter with an externally specified variable. To do so in this example, address 73 would have a POPC instruction.



ANDPUSHI constant GOTO labelA

This command pushes the eight inputs (I7 to I0) onto the stack. It is identical to the PUSHI GOTO labelA command, except that the inputs are first bit-wise ANDed with a constant to allow the masking of irrelevant inputs. If the GOTO instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is ANDPUSHI 34D GOTO labelA, where labelA is located at address 73. At the leading edge of the clock, the eight inputs are masked with the decimal constant 34 and pushed onto the stack. The next instruction comes from address 73. ANDPUSHI is an advanced instruction typically used to branch to an externally specified resource or to externally load the counter.



POPXORC constant GOTO labelA

This instruction pops the top-of-stack, bit-wise XORs it with a constant, loads the results into the counter, and jumps to labelA. If the GOTO instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is POPXORC 25D GOTO labelA, where labelA is located at address 73. The top-of-stack is popped off the stack, XORed with decimal 25, and the result is loaded into the counter. The next state comes from address 73. POPXORC is an advanced instruction typically used to compare the inputs against a known value and then branch on the basis of the result.

Table 1 summarizes the effects of each instruction on the address multiplexer, the stack, and the counter.

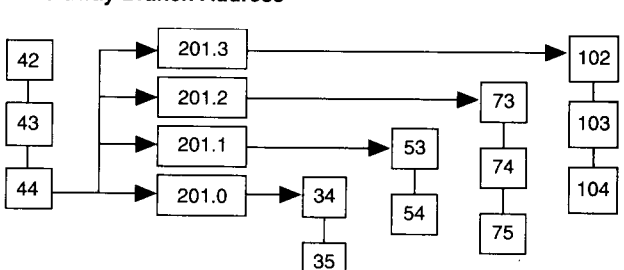
Table 1. Instruction Set Summary

Instruction	Definition	Next-State Address	Effect on Stack	Effect on Counter
CONTINUE	Continue with next instruction	labelA	None	Hold
JUMP	Jump to a label	labelA	None	Hold
CALL	Call subroutine	labelA	labelB	Hold
RETURN	Return from subroutine	labelA	Pop	Hold
LOADC	Load CREG	labelA	None	Constant
LOOPNZ	Loop/decrement on non-zero	labelA or labelB	None	Decrement
DECNZ	Decrement CREG on non-zero	labelA	None	Decrement
PUSHLOADC	Push CREG to stack and load CREG	labelA	CREG	Constant
POPC	Pop stack to CREG	labelA	Pop	Stack
PUSH	Push constant to stack	labelA	Push	Hold
PUSHI	Push inputs to stack	labelA	Inputs	Hold
ANDPUSHI	Push masked inputs to stack	labelA	Inputs (ANDed) Constant	Hold
POPXORC	XOR stack with constant and send result to CREG	labelA	Pop	Stack XOR Constant

Multiway Branching

Multiway branching provides an added dimension to the capabilities of the instruction set. For example, a JUMP labelA to an address within the multiway branch block forces the branch-select logic to decide which of the four words to send to the pipeline register. This selection is based on user-defined functions of the inputs. See Figure 10.

Figure 10. Jumping to a Multiway Branch Address



Any of the 13 available commands can be enhanced with multiway branching. For example, location 44 in Figure 10 can be a CALL to a subroutine, and address 201 can contain the starting instruction for 4 unique subroutines. The routine that is actually executed depends on the user-defined condition of the inputs. The following ASM code can be used to implement this example:

```
44D:    [Output Spec] CALL labelA;
201D:   IF        cond1     THEN   [out 1]    JUMP 102D;
          ELSEIF  cond2     THEN   [out 2]    JUMP 73D;
          ELSEIF  cond3     THEN   [out 3]    JUMP 53D;
          ELSE                     [out 4]    JUMP 34D;
```

Design Security

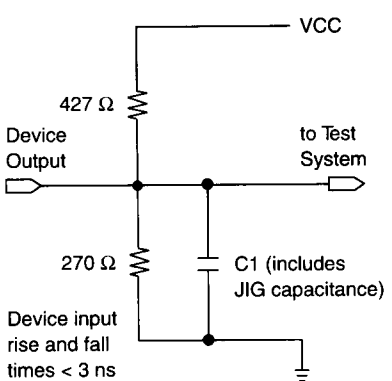
The EPS448 EPLD contains a programmable design Security Bit that controls access to the data programmed into the EPLD. If this Security Bit is used, a proprietary design implemented in the EPLD cannot be copied or retrieved. It provides a high level of design control because programmed data within EPROM cells is invisible. The Security Bit, along with all other program data, is reset by erasing the EPLD.

Functional Testing

The EPS448 EPLD is fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements, thus ensuring 100% programming yield. AC test measurements are performed under the conditions shown in Figure 11.

Figure 11. EPS448 AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large-amplitude, fast ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.



Since the EPS448 EPLD is erasable, Altera can use and then erase test programs during early stages of production flow. This ability to use application-independent, general-purpose tests is called generic testing and is unique among user-defined LSI logic devices. EPS448 EPLDs also contain on-board test circuitry to allow verification of function and AC specifications after they are packaged in windowless packages.

Figure 12 shows output drive characteristics for EPS448 I/O pins and typical supply current versus frequency for the EPS448 EPLD.

Figure 12. EPS448 Output Drive Characteristics and I_{CC} vs. Frequency

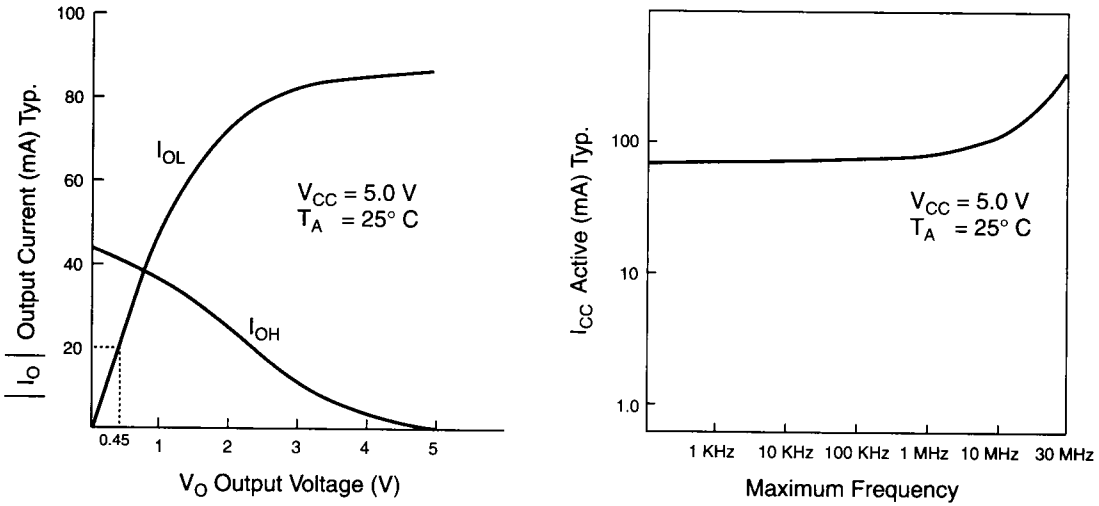
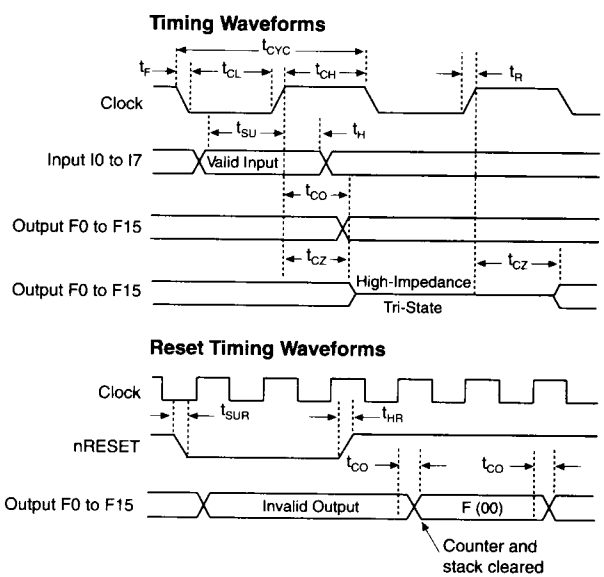


Figure 13 shows EPS448 timing and reset timing waveforms.

Figure 13. EPS448 Switching Waveforms

If nRESET is held low for more than three clock edges, then the outputs associated with the boot address (00 Hex) will remain at the pins until the third clock after nRESET goes high.



5 STG & SAM EPLDs

Absolute Maximum Ratings

Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	14.0	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–250	250	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			1200	mW
T_{STG}	Storage temperature	No bias	–65	150	° C
T_{AMB}	Ambient temperature	Under bias	–10	85	° C

Recommended Operating Conditions

See Note (2)

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	70	° C
T_A	Operating temperature	For industrial use	–40	85	° C
T_C	Case temperature	For military use	–55	125	° C
t_R	Input rise time			500 (100)	ns
t_F	Input fall time			500 (100)	ns

DC Operating Conditions

See Notes (2), (3), (4)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –8 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –4 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 8 (4) mA DC			0.45	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND, *See Note (5)*	–10		10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–10		10	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND, *See Note (6)*		60	95 (120)	mA
I_{CC3}	V_{CC} supply current (active)	No load, 50% duty cycle, f = 1.0 MHz		90	140 (200)	mA

Capacitance *See Note (7)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		15	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{RST}	nRESET pin capacitance			75	pF

AC Operating Conditions *See Note (3)*

			EPS448-30		EPS448-25		EPS448-20		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
f_{CYC}	Maximum frequency	C1 = 35 pF	30		25		20		MHz
t_{CYC}	Minimum clock cycle			33.3		40		50	ns
t_S	Input setup time		16.5		20		22		ns
t_H	Input hold time		0		0		0		ns
t_{CO}	Clock to output delay	C1 = 35 pF		16.5		20		22	ns
t_{CZ}	Clock to output disable or enable			16.5		20		22	ns
t_{CL}	Global clock low time		11		12		15		ns
t_{CH}	Global clock high time		11		12		15		ns
t_{SUR}	nRESET setup time		16.5		18		18		ns
t_{HR}	nRESET hold time		5		5		5		ns

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature versions.
(3) Operating conditions: V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use.
(4) Typical values are for T_A = 25° C, V_{CC} = 5 V.
(5) For 1.0 < V_I < 3.8, the nRESET pin will source up to 200 µA.
(6) This condition applies when the present state is a single-way branch location.
(7) Capacitance is measured at 25° C. Sample-tested only.

Product Availability

Operating Temperature		Availbility
Commercial	(0° C to 70° C)	EPS448-20, EPS448-25, EPS448-30
Industrial	(–40° C to 85° C)	EPS448-20
Military	(–55° C to 125° C)	EPS448-20

Note: Only military temperature-range EPLDs are listed above. MIL-STD-883B-compliant product specifications are provided in Military Product Drawings (MPDs), available by calling Altera's Marketing Department at (408) 984-2800. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.