

RELIANCE-1 DSP SYSTEM

Jeung Joon Lee. 2 June, 1998

1.0 SYSTEM FEATURES

- **0.625 MIPS peak sustained performance**
- **Integration of fixed-point DSP core with dual 11 bit ADC and dual 11 bit DAC for CD-quality audio signal processing**
- **Architecture supports *parallel* and/or *cascade* digital filter realizations with zero overhead.**
- **DSP core has 24 bit datapath supporting 12x12 bit signed multiplier and 24 bit adder**
- **Two stage pipeline allows a complete MAC operation, including opcode and operand fetches and write-back, in 5 cycles**
- **Fixed length instruction format for ease of programming**
- **128 program instructions and 128 data coefficient support**
- **Instruction format supports hardware parallelism**
- **User transparent boot-loading from EEPROM to SRAM upon system reset**
- **Internal clock speed of 3.125Mhz. Ratio of 1:8 from system clock to internal clock**
- **Prototype implementation on 7 Lattice ISP FPGA (3 ispLSI 3256, and 4 ispLSI 1016)**

2.0 GENERAL OVERVIEW

Reliance-1 is a general purpose stand-alone 12 Bit fixed-point Digital Signal Processor (DSP) system optimized for performing second order FIR digital filter as *transposed direct form II*. Higher order filters are possible with minor restrictions. Reliance-1 is composed of a 12 bit fixed-point DSP core, 11 bit dual ADC, 11 bit dual DAC, 8 bit flash EEPROM memory and 24 bit fast SRAM (20nS access time). Refer to figure 1 for system block diagram.

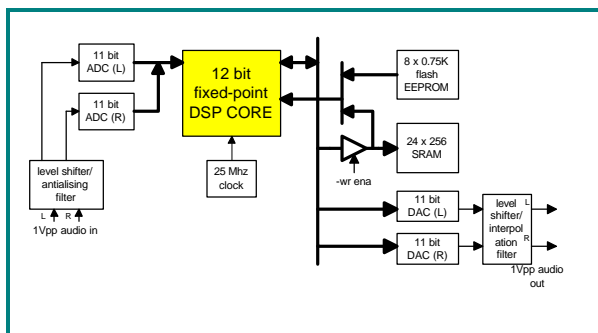


Figure 1 Reliance-1 stand-alone DSP System Block Diagram

The flash EEPROM contains the executable op-codes as well as filter coefficients or other user constants. The address space imposed by the DSP core is 1 Kbytes. The SRAM is the memory from which the DSP core executes. The maximum SRAM space imposed by the DSP core is 256 words (word = 3 bytes). The SRAM space is split into 'code' space and 'data' space, 128 words each. This allows a much more efficient instruction format. After power-up or system reset, the contents of the EEPROM are boot-loaded into the SRAM. This boot-loading is completely transparent to the user. This architecture allows the use of a high capacity slow access EEPROM to hold code and yet allow the use of high speed SRAM during execution for improved performance.

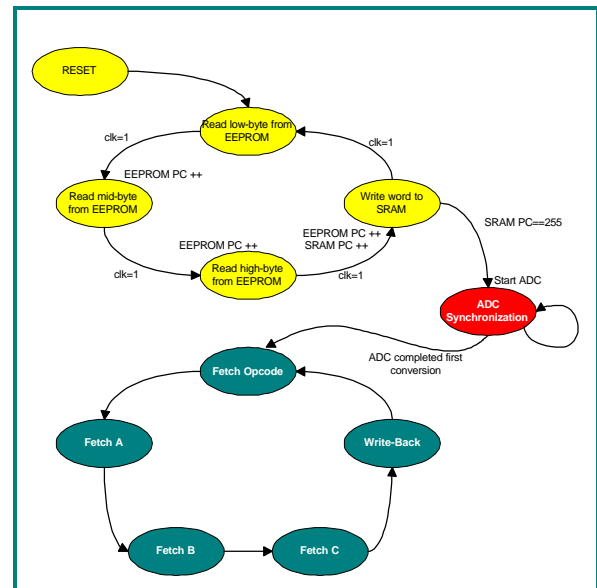


Figure 2 DSP Core's top-level state flow

The incorporation of dual 11-bit ADC and dual 11-bit DAC to the DSP core allows CD-quality stereo audio signal processing possible without compromises. The sampling rate is only limited to the program size. The fastest sampling rate is 312.5 KHz¹, corresponding to the execution of 1 instruction (plus 1 instruction delay for pipeline overhead), while the slowest sample rate is 4.845 KHz, corresponding to the execution of 128 instructions (plus 1 instruction delay for pipeline overhead). A typical second order FIR difference equation requires 7 instructions to complete (per channel), corresponding to 78.125Khz sampling rate (plus 1 instruction delay for pipeline overhead).

The DSP core runs at 1/8 of the oscillator clock, or at 3.125Mhz, giving a clock period of 320nS. The instruction (i.e. op-code) width is fixed to 21 bits and supports *hardware parallelism*. The instruction format and the datapath has been

¹ The top sampling rate of 312.5Khz is only theoretical, and cannot be achieved due to the limitation of the ADC used in the system. Reliance-1 uses Burr-Brown's ADS7804 which guarantees 100Khz peak sample rate.

optimized to perform *multiply and accumulate* (MAC) operation, along with the op-code and appropriate operand fetches and result write-back all in one instruction cycle. One instruction cycle is composed of 10 system clock periods ('system states' or 'states' for further references), or 3.2uS. A two stage pipeline architecture maintains 100% throughput on the multiplier and allows completion of instruction execution

Due to the instruction optimization, operand A takes on the filter coefficient value read from the SRAM. Operand B takes on the input sample value or the result of previous operation, both of which comes from the internal register file. Operand C takes on the intermediate values (delay taps) read from the SRAM. Result R can be written to the internal register file or

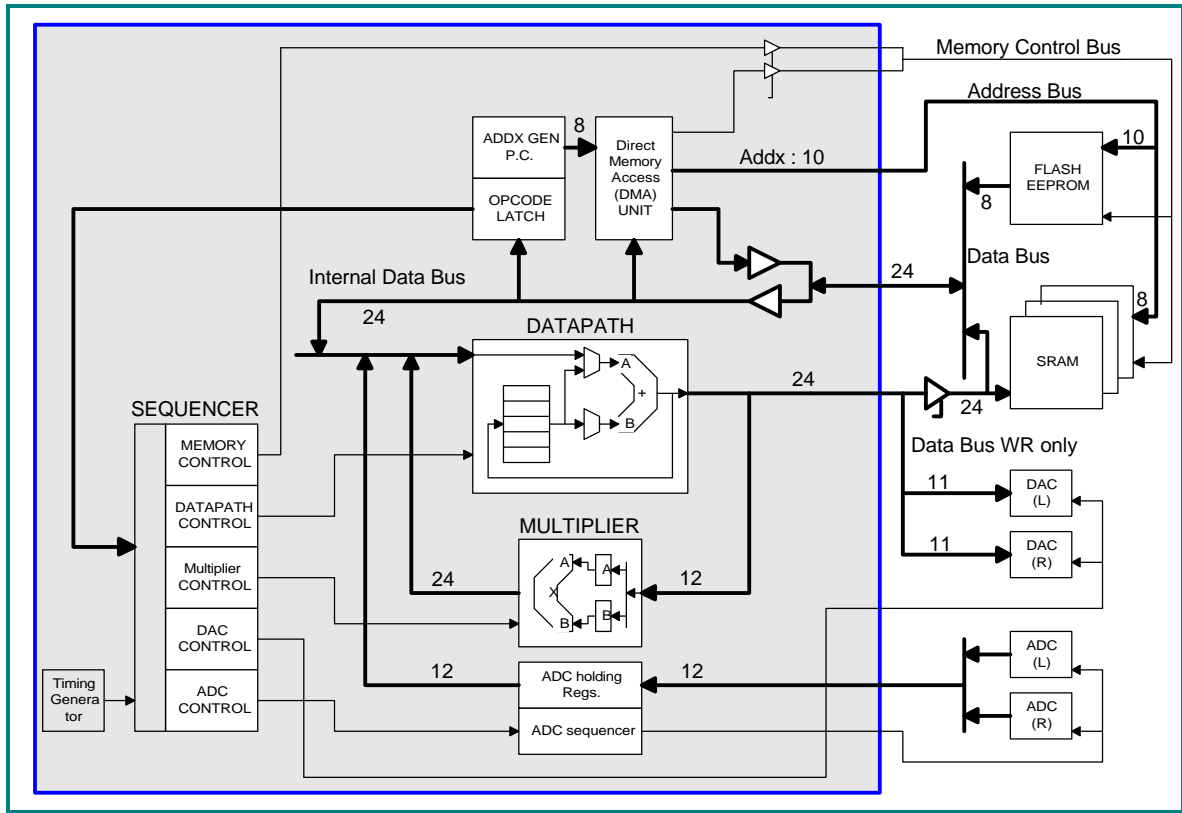


Figure 3 DSP CORE Functional Block Diagram. Not Reflective of Actual FPGA Implementation

every 5 states. The multiplier is capable of completing one 12 bit by 12 bit multiplication in four states. However, an additional states is needed as overhead, requiring 5 states total. The multiplier is fundamentally unsigned, but sign monitors, and sign inverters in the datapath allows user transparent signed multiplication in five states.

The datapath architecture allows efficient realizations of digital filters in cascade and parallel forms (in second order transposed direct form 2). This allows building of notch filters as cascade form and band pass filters as parallel forms with ease.

One instruction cycle is comprised of 5 distinct states: op-code fetch, operand A fetch, operand B fetch, operand C fetch and result write-back. Every instruction performs the operation

$$R = A \times B + C$$

to the SRAM. By letting $R = C$, recursive cumulative operations are possible. However *data hazards*, prevalent in pipelined architectures, must be observed.

3.0 FUNCTIONAL DSP CORE DESCRIPTION

Figure 2 illustrates the functional block diagram of the DSP Core. This block diagram is *not* reflective of the actual implementation, but is convenient in illustrating the functional units making up the DSP Core. Seven FPGA packages comprises the Core, and some of the buses are replicated throughout the various FPGA packages, necessitated by the routing and fitting effort.

The DSP Core is comprised of 7 distinct functional units: (1) 24 bit wide datapath, (2) 12x12 unsigned multiplier, (3)

sequencer, (4) timing generator, (5) DMA unit, (6) address generator and op-code latch, and (7) ADC interface. The datapath and the multiplier (unsigned by nature) comprises the core of the numerical processing power. Although the multiplier is limited to 12 bit operands, the datapath is able to accommodate 24 wide results, and is thus able to provide a limited amount of overflow protection. This is critical in operations needing accumulation, where overflow is to be avoided to prevent undesirable effects such as audio 'clicks' (as will be presented in section 3.1, Reliance-1 provide no over or under flow protection). The datapath and the multiplier is housed in separate large capacity FPGA package (Lattice Sem. isp3256. Capacity 11K gates).

The sequencer is the realization of the behavioral algorithm of the DSP Core. Taking the opcode and the clock signals as inputs, the sequencer is responsible for managing, the proper and desired, flow of data among the various internal units and the external peripherals (SRAM, DACs and ADCs). With the exception of a few key signals needing pipeline support, the sequencer is composed of combinatorial gates. Three FPGA packages houses the sequencer. The choice of the package (Lattice Sem. isp1016. Capacity 2K gates) was based on the speed rather than capacity, since the sequencing effort needs to have as little time delay as possible.

The timing generator is responsible for synthesizing the various clock signals, including the instruction state counter needed by the DSP core. Three major type of clock signals are provided to the DSP core: (1) 50% duty primary system clock, (2) 25% duty pulse clock, and (3) state identification signals. The timing generator is housed in Lattice isp1016 package.

The DMA unit (direct memory access) is responsible for the boot-loading from the EEPROM into the SRAM. This boot-loading occurs after every system reset (including power-on reset). Three bytes are read from the EEPROM and written to SRAM as a word (word = 3 bytes). The relative ordering of bytes in EEPROM is *big-endian*. That is, the most significant byte is stored in the lower order address. All of the required control signals as well as the data and address buses are provided by the DMA (as all DMA's do), and are tri-stated during the normal DSP Core operation to allow the bus controller gain access to the SRAM. During the DMA transfers, the clock speed is reduced in order to allow ample time for the slow EEPROM access (100 to 150ns). The DMA transfer ends when 768 bytes have been read from the EEPROM to SRAM. At this time, the ADC synchronizer is enabled. This is required to guarantee that ADC samples are available when the first op-code executes.

The address generator is responsible for maintaining the code space program counter (PC) as well as the data space coefficient pointer. During the opcode fetch state (first state) the address of the opcode to be fetched is presented to SRAM. During the remaining states, the addresses of the location of the coefficients, derived from the opcode, are presented to the SRAM. During the boot-load process, the PC is used to point to the destination address of the transfers. In the same

functional block, the op-code fetch latch resides. This is desirable since a significant SRAM accesses are dictated by the addresses derived from the opcode. The opcode latch contains pipe-latches wherever required.

The ADC interface contains holding registers as well as timing generators. The holding registers allows the ADC's to perform the next conversion whenever the current conversion is completed. The particular ADC's used by the Reliance-1 (Burr-Brown ADS7804) does not contain internal latches. The timing generators provide the necessary conversion and data transfer signals to the ADC. This allows both ADC's (for left and right channels) to be read in one state as well as generating the 'conversion-start' signals synchronously.

3.1 Datapath

The datapath and the multiplier comprises the heart of Reliance-1's DSP Core. Although it is customary to consider the multiplier and the datapath as a single entity, Reliance-1 treat them as separate. This is due to the fact that conceptually, they perform two very distinct functions, and in Reliance-1's context, these units operate in parallel.

Within the datapath, four major blocks are visible: (1) 24 bit ripple full-adder, (2) 24 bit inverter, (3) 24 bit multiplexer and (4) register file. This datapath differs somewhat from the classical datapath, in one respect: the lack of a formal arithmetic logic unit (ALU). A direct consequence of this is the inability to perform conditional operations (most notably, branching). Reliance-1 is a highly optimized finite state machine designed to implement digital filters in second order units, which does not require a full fledged ALU (nor conditional operations). Two ALU primitives are supported, however: logical inversion (one's complement), and limited shifting capability. Both functions are essential in the filtering operations. The former is needed as part of sign inversion for the (unsigned) multiplier. The latter is used to 'divide' the result of the filter by the large coefficient multiplier. This shifter is available only to one register ('Y') in the register file.

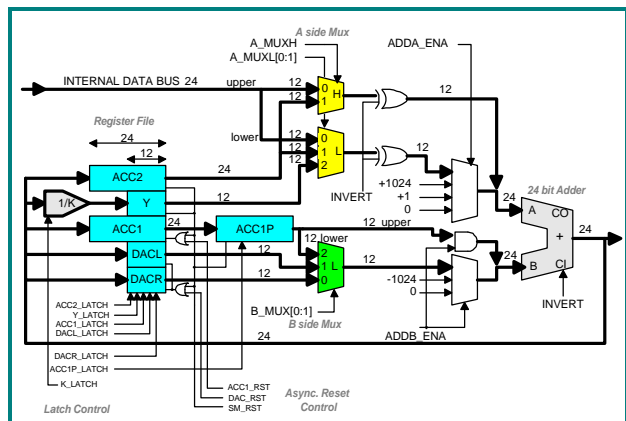


Figure 4 Functional Block Diagram of Reliance-1's internal Datapath

The adder consists of simple full 2 bit adders extended 24 times. Although the speed performance of this kind of adder is usually inferior to the carry look-ahead type, FPGA real-estate was a major contributor in the decision process. The carry out of the adder is a convenient feedback signal to drive overflow and underflow protection units, but real estate restriction prevented the existence of such unit in Reliance-1.

The 24 bit inverter is placed between the adder and the A side multiplexer. This placement allows sign monitoring on coefficients, input samples and results of previous multiplication operations. Sign monitoring and sign inversion is needed in order to perform signed multiplication on a unsigned multiplier.

Both inputs of the 24 bit adder are connected to a multiplexer which routes the appropriate operands (or data), dictated by the op-code, to the adder. The A side path contains two 12 bit muxes, one for the low order half and one for the higher order half. The low order half mux can select from internal registers ACC2, Y or from the internal bus. From figure 2, note that the internal bus (24 bit wide) has access to the SRAM, output of the multiplier and the output of the ADC holding registers. The high order mux can select from the internal bus and ACC2. On the B side path, only one 12 bit mux is available and can select from ACC1P, DACL and DACR, all internal registers.

Six registers comprises the register file in the datapath: Y, ACC1, ACC1P, ACC2, DACL and DACR. Out of these 6 registers, ACC1P and ACC2 can not be used as result destination of a MAC operation. The registers are constructed of positive clock edge D-flipflops with active-high asynchronous resets.

- **Register Y** is 12 bits wide and is intended to be used to store the final result of a second order FIR filter. For this reason, in the input path of this register a 4 setting divider is available. The divider settings are: 1, 16, 64 and 512. The divider setting is 1 after a system reset. In a cascade filter structure, this register can be used to provide the input to the next filter stage.
- **Registers ACC1 and ACC1P** are both 24 bit wide, and are used by the sequencer to store the 'C' operand fetched from the SRAM. Register ACC1P is the pipe-register, and is updated from ACC1 during state 4 (fetch C). Due to the pipeline architecture and instruction optimization, this is the only register needing a pipe-register. Potential *read-before-write data hazards* can occur if two instructions back to back are referencing to this register. However, there is one exception to this the hardware allows, which was implemented in order to improve processing throughput. Refer to section 4.
- **Register ACC2** is a 24 bit register and is used for internal housekeeping, particularly during state 3 (fetch B) and state 5 (write back). This register assists in sign inversion of multiplier result and level shifting of the input sample. This register has no visibility to the op-code, and thus to the programmer.

- **Registers DACL and DACR** are both 12 bits wide and are used to hold the data to be sent to the DACs. No other registers or means exists to update the DACs. Due to the hardware parallelism, these registers can be selected as destination of a MAC operation, *in conjunction* to Y, ACC1 or SRAM. However only DACL or DACR is allowed in this operation.

3.2 Multiplier

The multiplier in the DSP Core operates on two 12 bit wide operands, producing 24bit result. The multiplier is unsigned, but with support from the datapath, signed multiplication is possible. Sign monitoring logic in the datapath/sequencer keeps track of the operand signs. When a negative operand is detected the sign inversion logic inverts it into positive (if the operands are positive, no inversion are done). The result of the multiplication will be sign inverted (i.e. made into negative) only if the logical XOR of the sign bit of both operand is true (i.e. only one of the operand is negative). This sign monitoring and inversion is handled by the sequencer/datapath and is transparent to the op-code.

The architectural implementation of the multiplier is *shift-and-add*. This is a much more conservative implementation (in terms of throughput performance, i.e. ops/second) as compared to algorithms like Karatsuba-Olfman. However, recursive algorithms allow a sound throughput versus circuit size ratio. Shift-and-add algorithms require N recursions of shift-add for N-bit by N-bit multiplication (excluding any overhead). In order to improve speed throughput, Reliance-1's multiplier replicates the shift-and-add logic three times. This allows a threefold speed improvement, by letting the three multiplier operate in parallel. Refer to figure 4 for functional block diagram. The resulting throughput is 4 clock-cycles per multiplication. Due to the finite delays (mostly synchronization delays) in the operand latching as well as result latching an extra cycle of overhead is required, producing a 5 cycle multiplier.

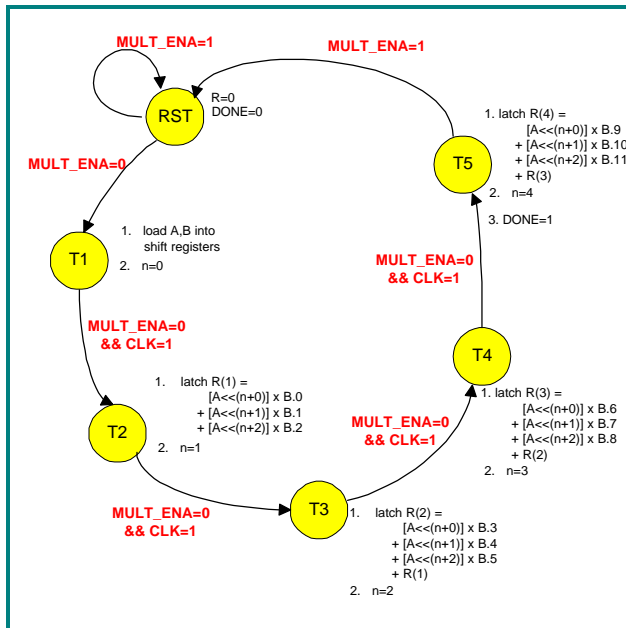


Figure 5 High level functional state diagram of the multiplier

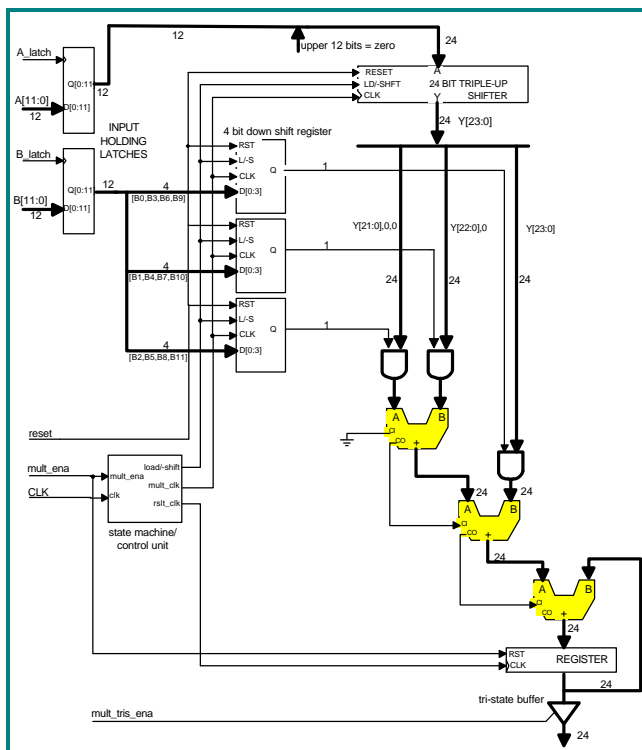


Figure 6 Functional Block Diagram of Reliance-1's 5 cycle Multiplier. Notice the triple shift-add architecture.

The multiplier has double buffered input latches, consisted by the input holding latches and latches internal to the shift registers. This effectively acts as pipe-latches, allowing the multiplier to load next operands while proceeding with current operation.

MULT_ENA is the primary control signal. When this signal is active high the output register is flushed (i.e. cleared) and the multiplier is held at 'stop' state. MULT_ENA and all other control signals are synchronous to CLK. The high to low falling transition starts the multiplication process. At the detection of this edge (coincident to rising edge of the clock), an internal signal CLK_ENA is set high. This signal gates CLK to produce MULT_CLK. On the very first rising edge of MULT_CLK, operands A and B are latched into their respective shift registers from the input holding registers. Operand A latch into a 24-bit triple up parallel in, parallel out shift register. This shifter is constructed of 24 DFF's with input muxes to select from neighboring DFF outputs or from external input. The 12-bit operand A is loaded into the lower order 12 bit of the shift register; the upper 12 bits are grounded. On every subsequent rising edge of MULT_CLK, the shift register shifts operand A up by 3 bits.

Operand B latches into three separate 4 bit parallel in serial out shift registers. Operand B is spliced into the three shift registers as [B0,B3,B6,B9], [B1,B4,B7,B10] and [B2,B5,B8,B11], (relative ordering: LSB => MSB). On every subsequent rising edge of MULT_CLK, the shift register shifts down by one bit, with the result of each shift register feeding into a 24bit AND gate which acts as a binary multiplier. The result of the AND gates are added together with the previous cumulative sum. The result holding register is also clocked from a gated version of MULT_CLK called RSLT_CLK. On the fifth rising edge of CLK, the result holding register is available with the final data. The DONE signal also becomes active high, indicating the completion of multiplication operation.

As seen in figure 5, once the multiplication is completed, (state T5), the control signal MULT_ENA, must be cycled in order to proceed with the next operation. The final result becomes available at the beginning of T5 (with some finite latency, of course), and the sequencer/datapath reads the result during the first half of T5 cycle. The second half of T5, MULT_ENA is cycled and the next multiplication started. Due to the dual stage pipeline architecture of the DSP core, the multiplier is kept at 100% usage (i.e. throughput). The first pipe stage consists of opcode and operand fetches (1 opcode and 3 operands A,B and C) and result write-back. The second stage consists of multiply and accumulate (MAC) operation. Both of the pipe-stages requires 5 cycles. In this regard, this pipe-line architecture deviates from the traditional sense of one system-clock cycle per pipe-stage. Alternatively, Reliance-1's dual pipe stage can be viewed as dual pipe-stage with each stage consisting of 5 sub-cycles.

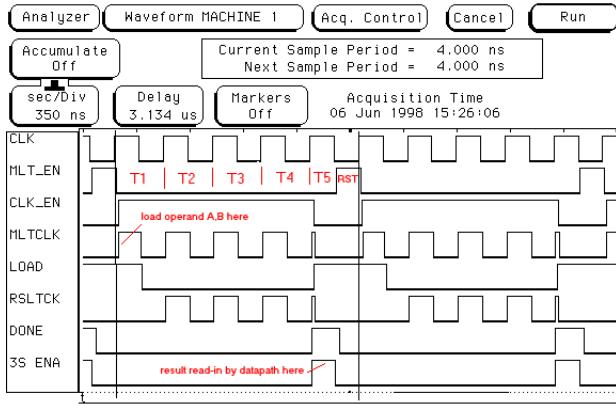


Figure 7 Actual measured Multiplier Timing diagram, showing two complete multiplier cycle, and other major timing signals

3.3 Timing Generator

The timing generator supplies all of the needed clocking signals to the sequencer, all of which are synthesized from the external clock source. The timing generator provides three major signals:

- 3.125Mhz 50% duty cycle primary system clock, CLK/2
- 6.250Mhz 25% duty cycle pulse clock, CLK4/4
- State identification signals:

SMC_ONE_HH,	SMC_ONE_LL
SMC_TWO_HH,	SMC_TWO_LL
SMC_THREE_HH,	SMC_THREE_LL
SMC_FOUR_HH,	SMC_FOUR_LL
SMC_FIVE_HH,	SMC_FIVE_LL

CLK/2 and CLK4/4 are generated from a synchronous up counter, and thus have the least propagation delay relative to each other, but does contain a modest delay relative to the external clock source.

CLK/2 is the main system clock and all events in the system are measured relative to it. The frequency of this clock is $1/8^{\text{th}}$ of the external clock source, or 3.125Mhz. This rate was empirically found to be most optimum speed in the given FPGA implementation.

CLK4/4 is a 25% duty cycle pulse signal used mainly to latch the various registers in the system, including the register file.

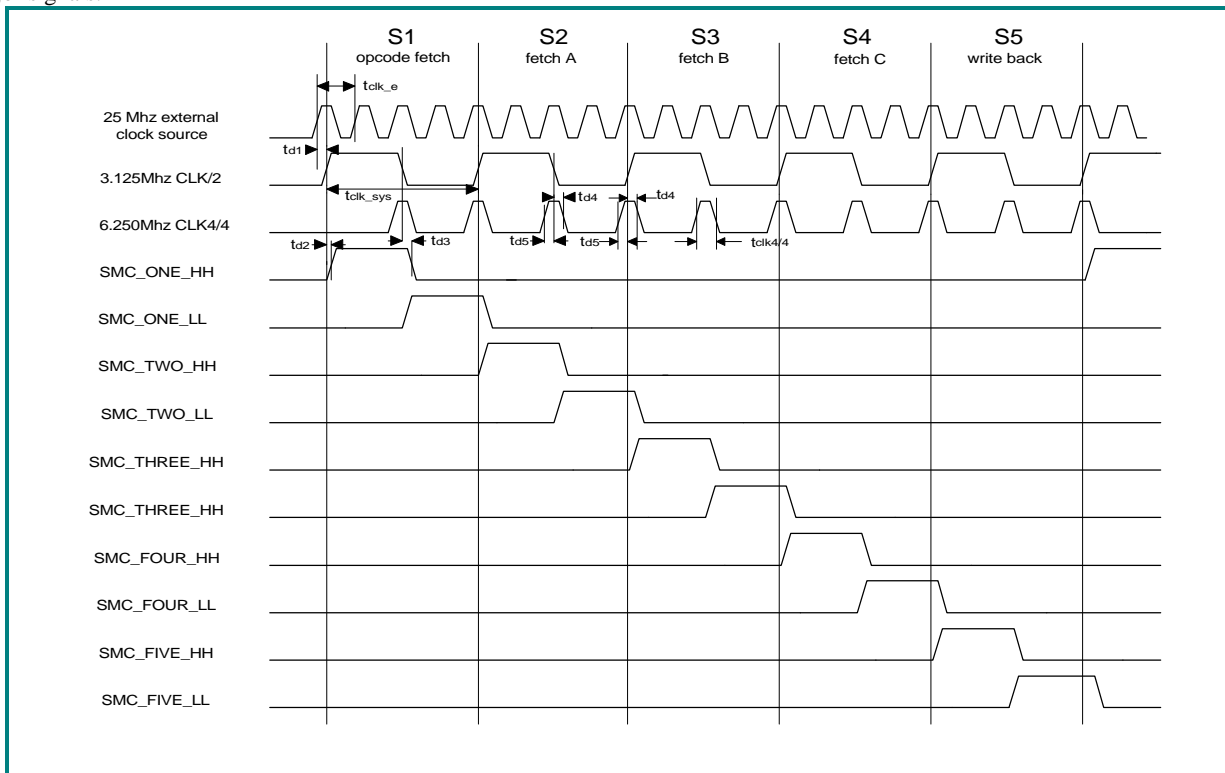


Figure 8 Major Signals provided by the Timing Generator

The advantage this signal provides is a finite and positive t_{d5} . Refer to figure 7. In many circumstances it is desirable to latch the register file before the rising edge of the CLK/2. An example of this is during SRAM read, where the read enable

signal is derived from CLK/2 (usually gated with the state identification and low portion of CLK/2, i.e.:

$$-OE = \text{NAND}(\sim\text{CLK}/2, \text{state_id}, \text{rd_control_gate})$$

Prior to the de-assertion of the read enable, it is desired to latch the output of SRAM to datapath's register file. This requires a clocking signal whose low-to-high transition occurs prior to the rising edge of CLK/2, and CLK4/4 provides this function.

There are 10 state identification signals provided by the Timing Generator, two signals in each of the five instruction states. The sequencer needs these id signals in order to carry out state-bound events. Due to this reason, it is critically important that the propagation delay, t_{d2} in figure 7, is as minimized as possible (1 gate delay or less). The timing generator achieves this, not by having a combinatorial logic at the output of a state counter, but by having two state counters. In the single state counter method, the counter would be clocked by positive edge of CLK/2, and the state id's would be decoded by a set of combinatorial gates. This introduces two-stage delay from the edge of CLK/2 to the state id: counter prop delay and the combinatorial gate prop delay. In a dual state counter method, one counter would be clocked by the rising edge of CLK/2 while the second counter would be clock by the falling edge of CLK/2. The first is responsible for the high (_HH) state id's and the latter for the low side (_LL). In this fashion, the state counters would be advanced one-half CLK/2 period earlier, providing to the state id outputs only with the propagation delay of the combinatorial gates.

The timing generator provides the 10 fully decoded state identification signals (at the expense of pin counts) rather than a 3 bit state counter output. This eliminates the need for combinatorial logics to decode the state counter's output, thus improving the propagation delay.

The decision to generate the fully decoded state identification signals was based on the need to reduce the propagation delay from the state counter to the gates using the

3.4 Sequencer

The sequencer in Reliance-1 is responsible for coordinating the activities of the various functional sub-blocks within the DSP Core. This coordination is essential in order to carry out a behavioral algorithm uniquely describing Reliance-1. All of the control signals (i.e. sequencing effort) are derived from the opcode fetched at the start of the instruction cycle and clock signals from the timing generator. The sequencer controls the datapath, multiplier, the external ADC, the external DAC, the SRAM and the address generator. The direct memory access unit (DMA) is the only block the sequencer does not control.

There are 21 control signals the sequencer uses to control the datapath. Refer to figure 4 for the datapath block diagram. These 21 signals can be grouped into categories by their function:

- Register file latching and resetting control signals

- Multiplexer selection control signals
- Multiplier sign support signals

Register file Latching Control	
ACC2_LATCH	AND(CLK4/4, NAND(-smc_three_hh, -smc_five_hh))
Y_LATCH	AND(opl20, opl0, opl1, smc_five_ll, CLK4/4)
ACC1_LATCH	AND(CLK4/4, NAND(NAND(-opl4, smc_four_ll) , NAND(-opl0, opl1, opl20, smc_five_ll)))
ACC1P_LATCH	NAND(pipe_ena, CLK4/4, smc_four_hh)
DACL_LATCH	AND(CLK4/4, OR(AND(-opl2, opl19, smc_five_ll, AND(cum, smc_one_hh))))
DACL_LATCH	AND(CLK4/4, OR(AND(opl2, opl19, smc_five_ll, AND(cum, smc_one_ll))))
K_LATCH	AND(smc_two_ll, CLK4/4, opl7, opl8, opl9)
CUM	DFF(q=cum, clk=smc_three, d=opl17, reset=OR(smc_two, sm_rst))
PIPE_ENA	DFF(q=pipe_ena, clk=smc_five, d=1, reset=sm_rst)
Opcode latch register OPL control	
OPL_LATCH	AND(CLK4/4, -CLK/2, smc_one)
OPLP_LATCH	AND(smc_one, -CLK/2)
Register File Resetting control	
ACC1_RST	AND(smc_four_ll, opl4, -opl3)
DSC_RST	AND(opl18, smc_two_hh)
Multiplexer Selection Control	
A_MUXH	OR(smc_three, smc_five_ll)
A_MUXL.0	A_MUXH
A_MUXL.1	OR(AND(smc_three_hh, -opl5, opl6), smc_one_hh, smc_one_ll)
B_MUXL.0	OR(AND(smc_one_hh, out_dac, AND(smc_one_hh, cum, -opl2, AND(smc_three, -opl2, opl5, opl6))))
B_MUXL.1	OR(smc_two_ll, smc_two_hh, smc_four_ll, smc_four_hh, smc_five_ll, smc_five_hh)
OUT_DAC	DFF_RST = AND(out_dac, smc_two) DFF(q=out_dac, clk=smc_five, d=xxyy, reset=ddf_rst) xxyy -> DFF(q=xxyy, clk=smc_five, d=opl14, reset=ddf_rst)

The multiplier requires 6 control signals:

- A_LATCH
- B_LATCH
- ~MULT_ENA
- MULT_ENA_TRIS
- CLK/2
- SM_RST

Multiplier Control Signals	
A_LATCH	AND(smc_two_ll, CLK4/4)
B_LATCH	AND(smc_three_ll, CLK4/4)
~MULT_ENA	smc_five_ll
MULT_ENA_TRIS	smc_five_hh
CLK/2	see text
SM_RST	system rest signal

Table 1 illustrates the sequencer's high level activity. Notice that the data bus and the datapath are the primary pipe stages (i.e. the dual pipe-line architecture targets to optimize the use of these two resources). Table 2 illustrates in more detail the control signals the sequencer uses to control the datapath and the multiplier.

In following the below state activity description, refer to the 'Basic Instruction Format' from figure 9.

SI: Opcode Fetch

Two major events occurs during this state:

1. During S1/High, the pipe-opcode latch, OPLP, is latched with the content of opcode latch, OPL. During S1/Low, OPL is latched with the 21 bit opcode read from SRAM.
2. If OPL14 (outdac) was set, two instructions ago, the digital to analog converters, left and right, are updated. During S1/High, the content of DACL register (from the register file) is added with 1024 and latched to the external left DAC. During S1/Low, the content of DACR register (from the register file) is added with 1024 and latched to the external right DAC.

Note from figure 3 that the presence of the tri-state buffer (output enable signal de-asserted during this time) in the data bus to the SRAM allows the simultaneous writing to the DAC's and the reading of the opcode from the SRAM.

S2: Fetch Operand A

During S2/Low state, the A operand as specified by the opcode (current opcode, just read during S1), is read from SRAM, 2's

complement performed if value is negative (sign bit saved), and latched into the A register in the multiplier.

S3: Fetch Operand B

During this state, the B operand as specified by the opcode (current opcode, just read during S1), is selected from one of the internal register file as follows:

During S3/High, the register (from the register file) selected as source of B operand is latched to register ACC2 (in the register file). If the source of operand B is from the ADCL/[R], -1024 is added to it.

During S3/Low, the content of register ACC2 is transferred to the operand B register in the multiplier. If the data is negative, two's complement is performed, and the sign bit saved.

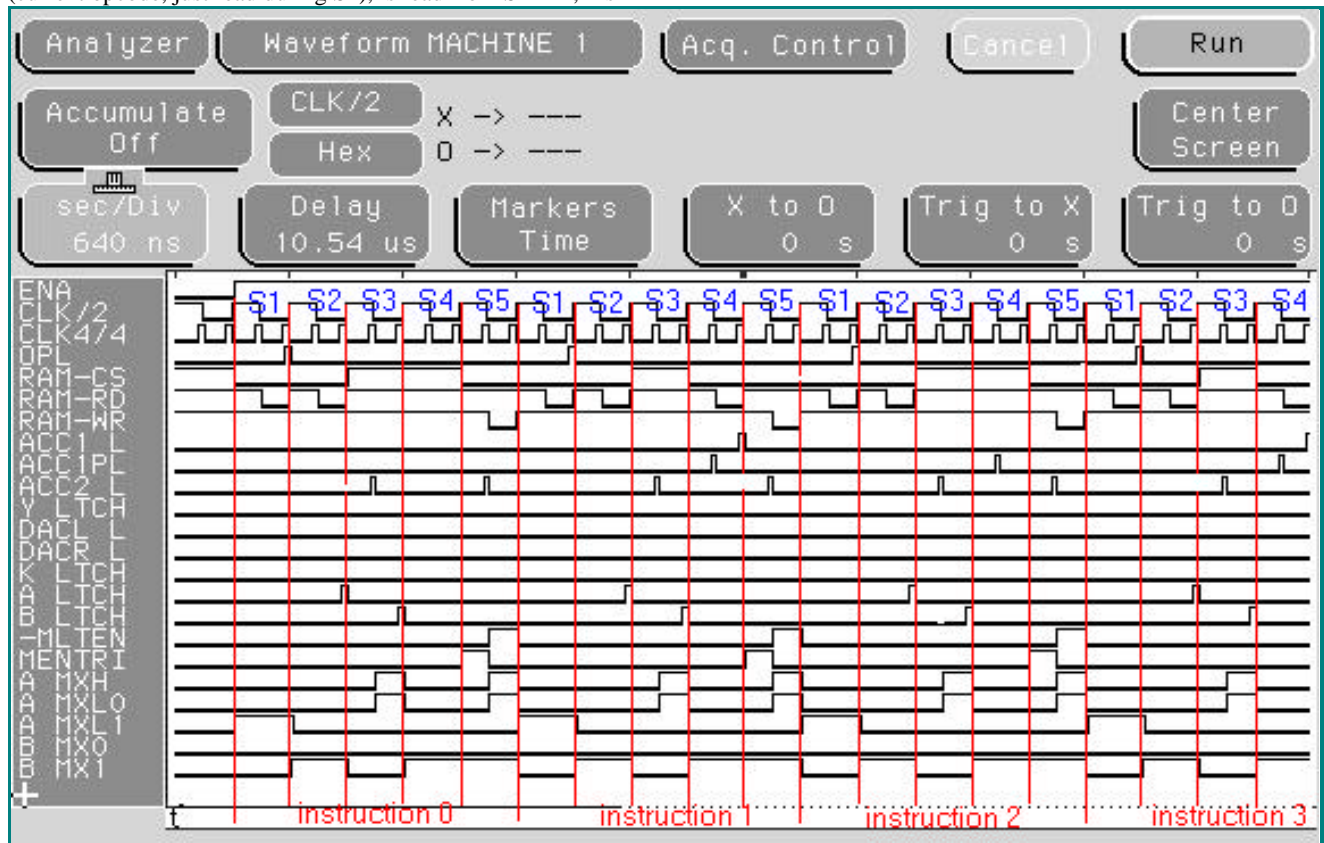


Figure 9 Sequencer's control signal on Datapath and the Multiplier.

S4: Fetch Operand C

During this state, the C operand, as specified by the opcode, (current opcode, just read during S1), is transferred to ACC1 register as follows:

During S4/High, the content of ACC1 is transferred to ACC1P, the pipe-register.

During S4/Low the source selected for C is latched to ACC1.

Note that 'C' can come from SRAM or internal register file.

S5: Write Back

During this state, the result of $A*B+C$ is written to the destination as specified by the opcode of the *previous* instruction, as follows:

S5/High, the multiplier is ready with the result of the previous multiplication. So read this into ACC2 register. If the signs (saved from previous instruction) differs, then allows the datapath to perform 2's complement.

S5/Low, the content of ACC2 is transferred to the destination specified by the previous instruction. Note that the destination can be to SRAM or to the internal register file.

Figure 9 illustrates the datapath and multiplier control signals during an actual program execution. Three complete instructions, executed after a system reset, are shown. The executed instructions are as follows:

1. $W1 = DACL \times 1 + 0$
2. $W2 = DACL \times 1 + W2$
3. $W1 = DACR \times 1 + 0$

3.5 Direct Memory Access (DMA) Unit

The DMA unit is responsible for 'copying' the content of EEPROM into the SRAM upon every system reset, including the power-on reset. Figure 10 illustrates the boot loading process. The state flow diagram is also illustrated in figure 2. Following the definition of a DMA, the DMA in Reliance-1's DSP core is a self-contained unit, and only needs the system reset SYS_RST input signal. All of the required signals such as address bus, data bus, control bus, source and destination address pointers and control units are integrated. The copying process ends when 256 words have been written to SRAM (i.e. 768 bytes read from EEPROM).

Figure 12 illustrates the functional block diagram of the DMA unit, notice the simplicity of the unit. The input holding latch, address counter and the control block comprises the heart of the unit. The SRAM address counter is part of the address generator, and is 'borrowed' during the DMA cycle.

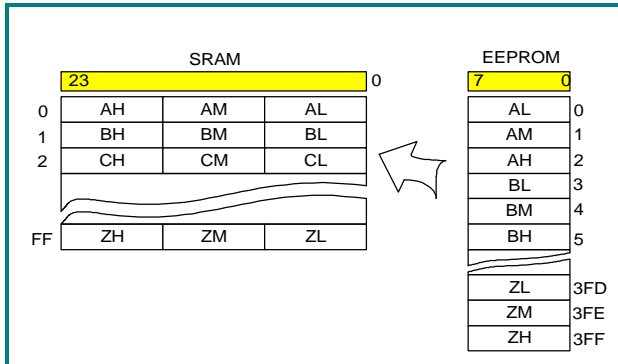


Figure 10 Boot-loading from EEPROM to SRAM occurs after every system reset.

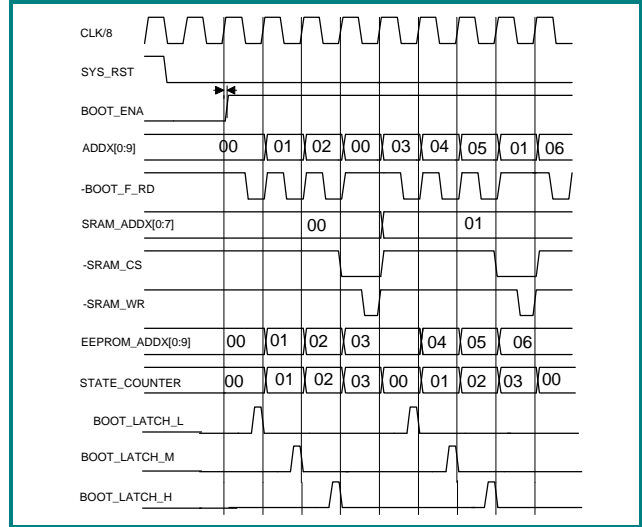


Figure 11 Major Timing Signals from the Boot Loader

Upon the detection of system reset (via reset button or power-up) BOOT_ENA is asserted high, signaling the boot load process is active. Refer to figure 11. Because the SYS_RST is an asynchronous signal, the synchronizer logic synchronizes BOOT_ENA to CLK/8 with a maximum delay of 2 clock cycles. -BOOT_ENA is a convenient signal to use as a -CS of the EEPROM. -BOOT_F_RD, the EEPROM's -RD signal is derived from the 2 bit state counter. -RAM_CS and RAM_WR are also derived from the state counter. Both of these signals are tristable and DMA gains access to SRAM only during the DMA cycle. The DMA performs 256 cycles of read-read-read-write operation. Three bytes are read from the EEPROM sequentially, and on the fourth cycle, the 24 bit data is written to SRAM. The entire DMA cycle takes $(256 * 4 + 2) * 16 * 320\text{nS} = 5.25 \mu\text{s}$. The source [address] pointer is a 10 bit up counter called EEPROM ADDX, which increments on every rising edge of EEPROM read signal -BOOT_F_RD. The destination [address] pointer is the same up-counter from the Address Generator Block. This is an 8 bit counter which, during the normal operation, is incremented by the state identification signal SMC_THREE. During the DMA cycle it is incremented by SRAM chip select, -SRAM_CS. The address multiplexer changes from source to destination [address] pointer on every fourth cycle of the read-read-read-write operation. When the destination address pointer changes from 0xFF to 0x00, the DMA cycle ends. BOOT_ENA returns to its inactive state (low) and a second signal BOOT_DONE becomes active (high).

Upon the end of DMA cycle, both of the A/D converters are commanded to initiate the digitization process. When the sampled data are available, the first instruction is fetched and executed. The purpose of this extra step is to guarantee that the first instruction will have a valid sampled data to operate on. Figure 2 illustrates the ADC SYNCHRONIZATION state.

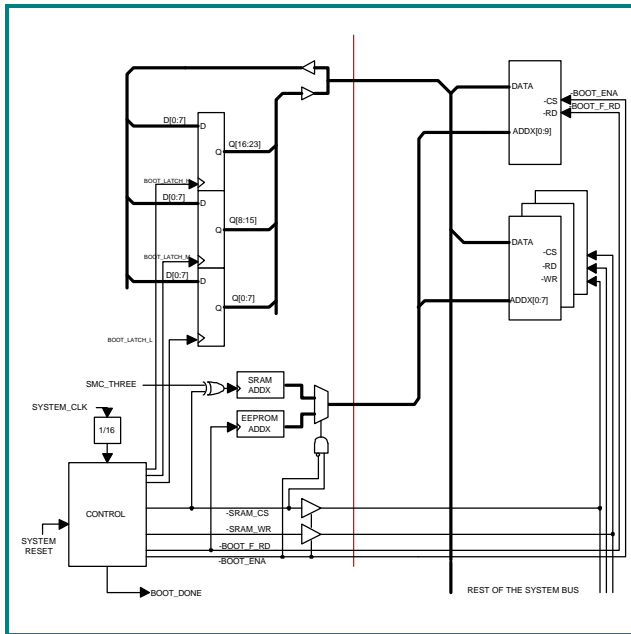


Figure 12 Functional Block Diagram of the Direct Memory Access (DMA) unit. This unit is responsible for the boot-loading of the EEPROM data into the SRAM

3.6 Address Generator

The address generator is responsible for maintaining the code program counter (PC), the opcode latch register OPL and generating the data address for reads and writes.

	OPCODE FETCH S1		FETCH A S2		FETCH B S3		FETCH W S4		WRITE BACK S5	
	High	Low	High	Low	High	Low	High	Low	High	Low
BUS ACTIVITY	update DAC's	fetch opcode (from sram)		fetch A (from sram) Latch A (to mult)		latch B (to mult)		fetch W (from sram) latch W (to reg file, reg ACC1)	fetch MULT (from multiplier)	write back (to SRAM)
DATA PATH	DACL= DACL+1024 or DACX= DACX + Y where X= 0= L 1= R	DACR = DACR+1024		A = -A + 1	ACC2 = sample - 1024 or ACC2 = Y or ACC2= DACx	B =-ACC2 + 1	ACC1P=ACC1	ACC1 = W	ACC2 = -MULT + 1	W1 or W2 or ACC1 or Y or DACL or DACR = ACC1P + ACC2

Table 1 Top Level Sequencer Activity

Parameter	Opcode Fetch S1		A Fetch S2		B Fetch S3		W Fetch S4		Write Back S5	
	H	L	H	L	H	L	H	L	H	L
A_MUXH	0		0 uncond.		0 opl6=0	1 uncond.	0		0	1
A_MUXL[0:1]	2		0 uncond		0 opl6=0 2 opl6=1 & opl5=0	1 uncond.	0		0	1
B_MUX[0:1]	1 out_dac=1 or cum=1 & opl2=0 0 cum=1 & opl2=1	0 out_dac=1 else 1	2 uncond.		1 opl[6:5] = [11] & opl2 = 0 0 opl[6:5] = [11] & opl2 = 1		2		2	
ADC_ENA_TRIS	0		0		1 opl6 = 0 else 0	0		0		0
MULT_ENA_TRIS	0		0		0		0		1	0
ONE	0		1 opl[9:7] = [000] else 0		0		0		0	
-1024	0		0		1 opl6=0 else 0	0	0		0	
+1024	1 out_dac = 1 else 0		0		0		0		0	
ADDA_ENA	1 out_dac = 1 else 0		0 opl[9:7] = [000] else 1		0 opl[6:5]=1 else 1	1	1		1	
ADDB_ENA	1 out_dac=1 or cum=1 else 0		0		1 opl[6:5]=1 else 0	0	0		0	1
INVERT	0		0	0 opl[9:7]='0' else macamx11	0	macamx11	0		mult_invert	0

Table 2 Multiplier and Datapath's major Sequencer Control signals

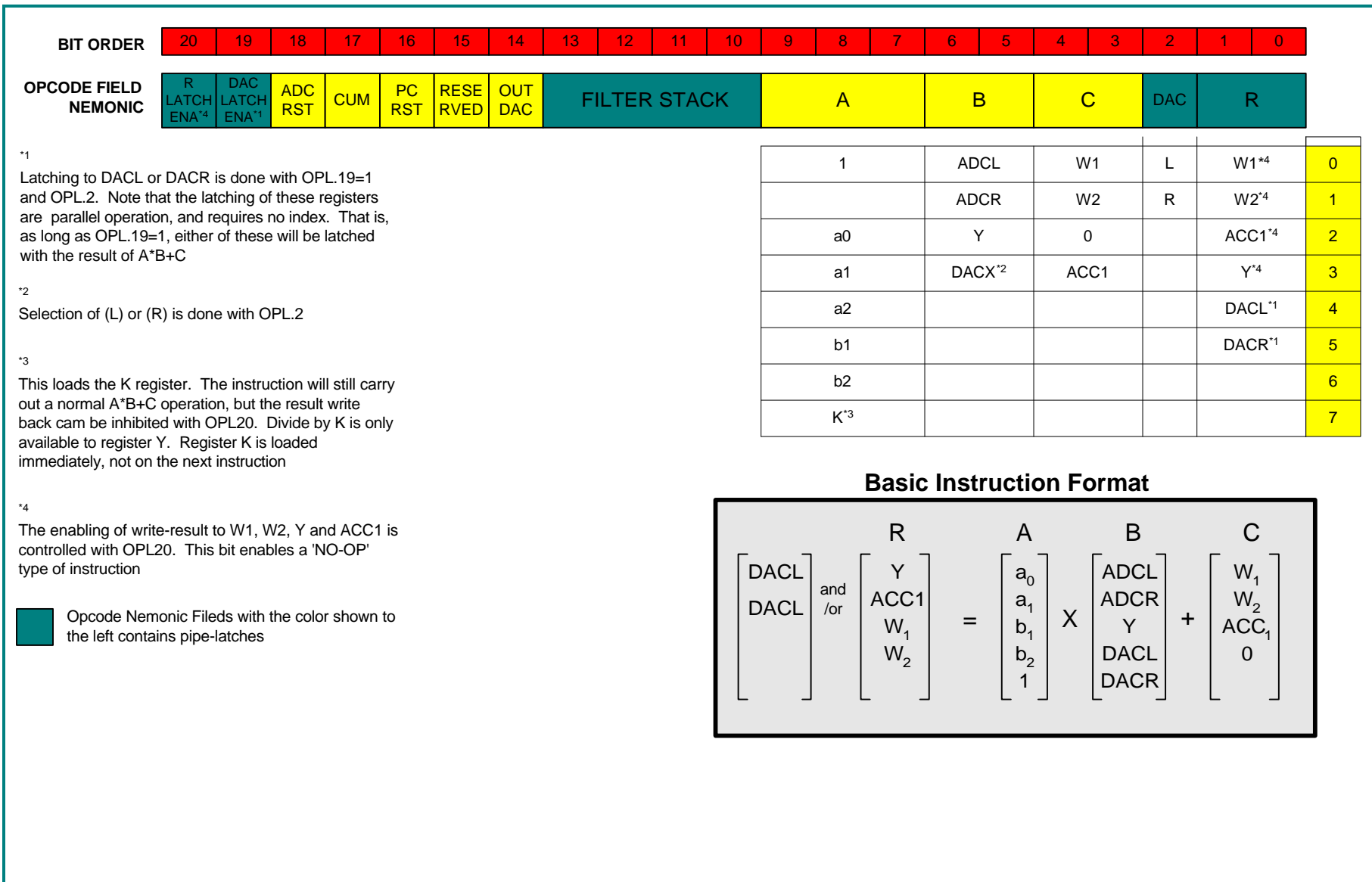


Figure 13 Opcode Format. Note the opcode supports only one instruction (see 'basic instruction format' box), yet can perform four tasks in parallel

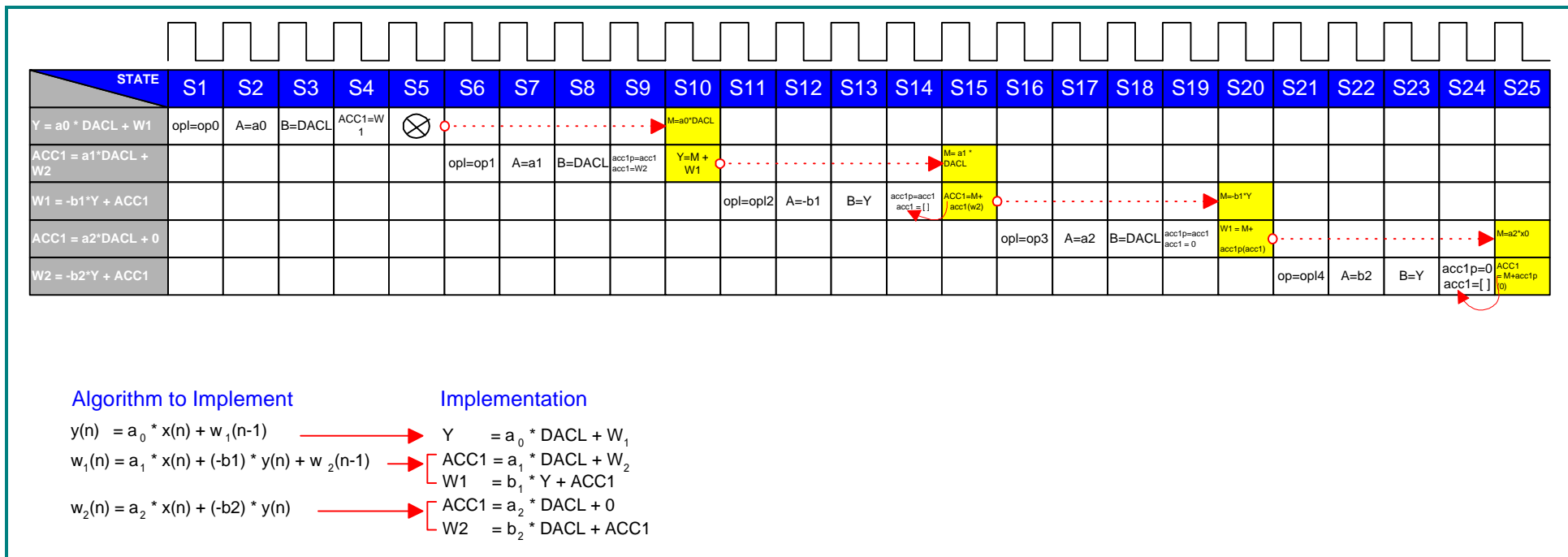


Figure 14 Illustration of a sample program: a second order filter realization in Transposed Form II

6.0 NEXT GENERATION

Reliance-2, the follow on to Reliance-1 will contain the following improvements:

1. External (peripheral) and internal (software) interrupt support
2. A dedicated arithmetic logic unit (ALU)
3. A more streamlined pipe-line architecture to allow instruction execution in fewer system states
4. Faster and more efficient multiplier
5. A more robust instruction format which allows:
 - NOP instruction
 - conditional and unconditional branchings
 - loop support
 - larger code and data space
6. Use of off-the-shelf integrated CODEC with serial interface such as TLC320AC02 to lower cost and package count.
7. Serial interface to facilitate code-down load during debug stage.
8. Full peripheral support to facilitate interfacing to LCD unit, keypad or keyboard units, etc.

Use the two tables below for the sequencer

Functional DSP core description

functional system description:

power-up sequences. reduced state flow

description of actions in each state

filter structure

Programming Description

→ mention data hazards. read before write.

=> Supported hardware parallelisms: dac update and DAC regs latching

=> discuss about the parallel and cascade filter structures.

Example of codes