Scan-chain Based Watch-points for Efficient Run-Time Debugging and Verification of FPGA Designs

Anurag Tiwari and Karen A. Tomko

Department of Electrical and Computer Engineering and Computer Science
University of Cincinnati
Cincinnati, OH 45221
E-mail: {atiwari, ktomko}@ececs.uc.edu

***Abstract*-- This paper describes a structured and area efficient approach for in-situ debugging of application for FPGA based reconfigurable systems. A scan chain is inserted into the hardware design running on the FPGA, which helps in debugging and verification by providing watch-point capability. The scan chain technique proposed is easy to use and has very low overhead. The scan-chain based implementation capitalizes on the capability of newer FPGAs to connect several LUTs serially and configure them as shift registers. The hardware debugging procedure proposed using the shift register LUTs does not require any recompilation of the design to change the watch-point conditions and thus, is very fast. In this paper the area overhead resulting from addition of a scan-chain based watch-point logic is discussed and is compared with other proposed debugging techniques. We observed that this technique has an average area overhead of 46% for the ITC benchmark circuits with varying widths of watch-point signals.**

1 Introduction

A typical reconfigurable computing application is made up of the hardware mapped on the FPGA device(s) present on a co-processor board and the software, which runs on the general-purpose processor. Debugging of these applications involves debugging of both the hardware and software components. Hardware simulation is one of the most widely used techniques for hardware debugging and validation. It allows the designer to examine the circuit in detail, but can be prohibitively slow. Large designs can take anywhere between a few hours to a few days for the complete simulation.

The problem of lengthy hardware debugging time can be mitigated by debugging the circuitry directly on the target platform. Debugging of reconfigurable computing applications on the target platform is possible, since the target platform is available before the application is completed. Some runtime debugging environments for FPGA are described in [1] [2] [3]. The key features behind these debugging efforts are the *readback* capability provided in some FPGAs [4] [5] along with the ability to stop and step the clock signal. A readback operation can acquire internal state of FPGA memory elements such as the LUTs, flip-flop and IOBs, which is then matched to the symbolic name in the original design. By stepping the clock to a cycle of interest before initiating readback, the user can analyze the values of the signals during execution. The drawback of readback is that the software overhead of a readback API call makes it a slow operation.

To overcome the slow speed of the readback operation, an additional *watch-point* circuit can be added into the design to reduce the frequency of readback. The added logic provides a designer with controllability, while the design executes at or close to the normal speed. The design running on the FPGA is executed until the watch-point condition occurs. The additional debugging circuit is removed from the design, when debugging and validation is completed. Thus, the final design has the same area and speed as the original design. The hardware watch-points therefore, enable a controlled execution of the hardware design and speed up the debugging procedure.

When the watch-point logic is added at the top most level in the design flow (i.e. in the VHDL design or in the netlist), it requires complete recompilation of the design for any change to a watch-point condition. Recompilation, which consists of synthesis, mapping and place and route of the design, is a time consuming procedure. The debugging process may require several iterations of watch-point logic modification while the error is being discovered and corrected in the circuit. However, the large recompilation time slows debugging. The addition of the watch-point logic mitigates debug time at the expense of area overhead; if this area overhead is large then the watch-point logic may not fit into the chip, as the capacity of an FPGA is fixed. In this paper, we have proposed a scan based watch-point logic technique which allows run-time change of the watch-point condition and is highly area efficient.

2 Related work

Addition of debugging logic in FPGA designs for debugging and validation purposes has been considered by other researchers in [6][7]. In [6] a traditional design level scan chain is proposed for debugging. However, average area overhead of this design chain is 84% and can reach up to 100%, which may restrict this technique to less congested designs. In our work a scan-chain is used to implement the watch condition logic. In [7], a technique to modify debugging logic is proposed using a java based design environment. This

technique limits designers to a java based structural design environment, which is less familiar than a behavioral HDL/Schematic environment. The technique proposed in [7] allows instrumenting the debugging logic at bitstream-level, but if modification is frequent, time to make the new bitstream and time to load the bitstream into the target FPGA may slow the debugging process. With support for partial reconfiguration the method in [7] will have similar overheads to the method presented in this paper which allows the user to change the watch condition without reconfiguring the FPGA.

There are a few commercial tools which provide automated and powerful features to add and modify the debugging logic in the design. Xilinx has a tool named Chipscope[8], which allows the designers to put embedded logic analyzer(ELA) cores into their designs. These ELAs can monitor design signals during design execution and can produce a trigger if the signals meet some predefined condition. The trigger conditions and signals monitored can be changed without any design recompilation. Chipscope needs a logic analyzer to view the signal status and a port on the reconfigurable computing board to connect it. In addition, the area overhead of the ELA is fixed, i.e. even if designer needs only a few signals to be monitored and requires only simple trigger conditions, the area overhead will still be large as a wide variety of capabilities are provided in the core. Altera also has a product named SignalTap[9], which is a logic analyzer embedded into the design running on the FPGA. SignalTap is similar to Chipscope in operation; however, any modification in the debugging logic except for changing the trigger condition requires complete recompilation of the design.

Validation and debugging of the design by adding debugging logic is not limited to FPGAs. For example, the Triscend E5 configurable system on chip platform [10] has on-chip debugging support using an additional breakpoint logic unit kept on the chip. This breakpoint unit monitors the user specified combinations of address and data control. The MCU freezes at the end of the current condition, whenever a breakpoint condition occurs. The breakpoint unit, though it aids the user in debugging, is limited to only the data, control and DMA signals. SIDSA also has a system on chip known as FIPSOC [11], which also has the hardware breakpoint capability [12]. The breakpoint mechanism in the FIPSOC is similar to that in Triscend E5, i.e. a breakpoint can be set only on user specified data and address values.

In [3], a software watch-point facility is presented in which the comparison between a user specified condition and the actual signal value is performed in software. This operation entails readback of the design signals at every clock cycle (single-stepping) or after a fixed number of clock cycles (multi-stepping). Single stepping the clock makes the debugging procedure very slow, as the software overhead of a readback operation is on the order of a second. On the other hand multi-stepping the clock may completely miss a user-desired event.

3 Reconfigurable Hardware Watch-points

As with hardware simulation, the addition of watch point logic allows any signal in the FPGA design to be watched for a particular value and condition. On every clock cycle, the user chosen design signals can be monitored for a specific condition. If there is a match between the signal value and the user specified watch-point condition, the design running on the FPGA stops executing and an interrupt is given to the application program running on a general-purpose computer (host). Upon getting an interrupt from the FPGA co-processor board, the software running on the host may initiate a readback operation to obtain the internal state of the circuit. The hardware execution cessation is achieved by disconnecting the clock used in the design. To provide the similar watch-point capability as software debugging tools, the design should be able to restart from the same point after the watch-point condition is reached. This requires control over the system clock, which should be disconnected from the design whenever a watch-point condition occurs, and should be connected back to design after the readback operation. This clock control is implemented using two Finite State Machines (FSMs) and a controllable clock buffer. An FSM takes input from watch-point logic implemented using the scan chain, when a watch-point condition is reached the FSM outputs an interrupt and stops the design clock. There is another control FSM which controls serial data shifting into the scan chain. Once the interrupt has been acknowledged by the software running on the general-purpose computer, the control FSM can enable the design clock. This operation is illustrated in figure 1.

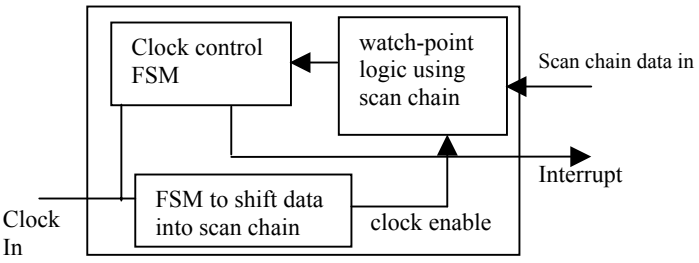

Fig 1: Diagram of clock control operation

We utilize Xilinx Virtex-II series of FPGAs in this research. Virtex-II FPGAs have low skew global clock buffers, which can be enabled by a control signal. We have used these controllable global clock buffers to stop the clock supply whenever there is an interrupt. The user can also use a gated clock methodology without such buffers to control the clock whenever there is a watch-point condition. Gated clock control methods are described in [13][14].

4 Watch-point logic implementation using scan chain

The hardware watch-points can be implemented by keeping a scan chain of LUTs in the design. There are some recently released FPGAs which allow the user to configure any LUT or RAM block as a shift register without using the flip-flops. The look-up table shift registers (srlut) has dual functionality, when in shift register mode it shifts the data into each of its memory locations. After the data is shifted into the srlut, it behaves as a normal LUT, i.e. producing an output based upon its memory location contents and its input address. Thus, if there is a 4 input LUT in the FPGA device, it can be configured as a 16 bit shift register without using any other logic. Altera's Stratix FPGA[15] and Xilinx's Virtex-II FPGAs[16] have the capability to configure any LUT as a shift register. Furthermore, individual shift register formed by LUTs can be combined together to form a bigger shift register. We call this shift register chain a scan chain of LUTs, since the appropriate data can be shifted into each and every bit of the shift register. This idea is analogous to the flip-flop scan chain in VLSI testing [17]. We take advantage of this concept for implementing hardware watch-points.

A variety of watch-point conditions can be set for monitoring a signal. Multiple watch-point conditions for a signal can be combined logically with "AND" or "Or" depending upon the users requirement. Similarly, if there are multiple signals with different trigger conditions, they can be combined similarly to make one interrupt output. Table 1 shows the flexible set of watch-point conditions implemented in our work.

Watch-point Condition	Description
Greater than	signal value is greater than the watch-point value
Greater than equal to	signal value is greater than or equal to the watch-point value
Less than	signal value is less than the watch-point value
Less than equal	signal value is less than or equal to the watch-point value
Not equal to	signal value is not equal to the watch-point value
Equal to	signal value is equal to the watch-point value
Rising edge only	signal makes rising edge transition
Falling edge only	signal makes falling edge transition
rising edge or falling edge	signal makes either a falling or rising edge transition

Table 1: Watch-point trigger conditions.

To implement hardware watch-points using srluts the design signals to be monitored are connected at the input address lines of LUTs. The srlut are programmed such that whenever a user desired condition on the signals connected at the input occurs, the output is asserted. In figure 2 there are three srluts connected serially, the input lines of which are connected to some design signals being monitored. The output lines of these three srluts are connected to the input lines of a fourth srlut, which can implement function such as "OR" or "And" etc depending upon the requirement. Furthermore, output of the fourth srlut is connected to the interrupt line to notify the host that watch-point condition has occurred.

The temporary register shown in figure 2 gets a watch-point condition specification from the host, and then transfers that data into the srluts. The maximum width of this register may be limited, for example it may be limited to 64 bits in a given co-processor board design, such as Annapolis Micro System's Wildstar. The watch-point data can either be transferred to it in parallel with 64 bits together or can be shifted from the host serially. If the data is transferred in parallel, and if there are more than 64 bits of watch-point signals in the design, the temporary register has to be loaded multiple times. When the watch-point data is transferred to the FPGA in parallel, a control state machine is kept in the design to sequence the parallel load and serial shift of data into the srluts. In addition, the temporary register in this case will be a parallel-in and serial-out shift register. It takes $16*N_{srlut}$ clock cycles to shift the data into all the srluts, where N_{srlut} is equal to number of srluts used to implement watch-point logic. The area overhead of the control state machine and the shift register can be reduced by transferring watch-point data serially into the FPGA. The serial shifting of watch-point data requires clock suspension and clock stepping support in the co-processors.

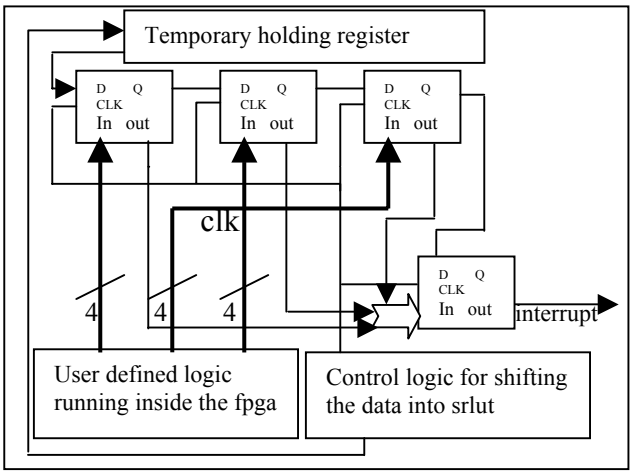

Fig 2: watch-point implementation using scan chain

Figure 3 shows the pseudo code for transferring watch-point data serially from the host using the co-processor board Application Programming Interface (API).

Procedure change_watch_point_values( )
{
1. ClkSuspend(BOOLEAN enable);

```
2.  for(int I=0;I<=number_of_watch_point_bits;i++)
3.  {  WritePeReg(int PeNum, data watch_point_bit);
4.     ClkStep( );
5.  }
6.  ClkFreeRun( );
}
```
Fig 3: Software controlled serial watch-point data shifting

4.1 Initialization and data shift into srluts

The srluts memory locations are initialized by the setting their attribute values in the VHDL file. This is helpful when there are only a few srluts in the design and only a few signal connected to them. For example, if there are only three signals connected to an srlut, only eight bits of data need to be shifted into it assuming that it is the last or the only srlut in the scan chain. Whenever a user wants to change the watch point condition, a control signal "*start_shift* " is asserted to the design and appropriate condition specification is given at the register input port. The *start_shift* signal can be a single bit of the register kept in the design. A control FSM is added into the design to synchronize the operation of shifting of data and then enabling the global clock control buffers. Upon receiving the *start_shift* signal, the controller starts shifting the data serially across the register chain. The advantage of this methodology of scan chain is that user can change the watch point signal at run-time simply by asserting the *start_shift* signal, which can be asserted using software API calls. Most importantly chip configuration and the time consuming synthesis, place and route process is bypassed completely using this technique.

5 Programming the SRLUT chain

The srluts can have their 4 bit input address lines connected to upto four different signals. Thus, any function made up of the four different design signals can be implemented inside the srlut. Consider a 4 bit signal A(0 to 3). If we implement a watch-point condition which produces an interrupt anytime the signal lines A(0 to 3) have a value that is greater than "1100" then depending upon how the A(0) to A(3) lines are connected to the address lines of the srlut, we implement this condition by programming the memory locations of the srlut. All the memory location addressable by signal A="1100" to A="1111" are programmed with a '1' and all other memory locations with a '0'. Now, whenever there is a condition with A>"1100" the srlut output will be logic '1' and an active high interrupt will be asserted. Similarly, for a watch condition where A="1100", only the memory location which corresponds to address "1100" will be programmed with logic value '1' and all other memory locations store logic value '0'. This process is applied to implement the all of the watch-point conditions mentioned earlier in table 1.

The binary stream, which is shifted into the srluts can be generated using a 'C' program developed as part of this research. The program takes as input the watch condition implemented by the srluts, and the order in which the signal lines to monitor are connected to the LUT inputs and generates a srlut bit-stream. When there are multiple srluts connected in series, the order in which the srluts are chained is also given to this 'C' program to generate the concatenated stream of bits for all of the srluts in the scan chain.

6 Interactive SRLUT Chain Programming

The scan chain technique can be easily integrated with an interactive GUI-based hardware/software co-debugging utility, such as developed in [3]. The reason behind quick integration of this technique with co-debugging is that the user only has to provide an expression for the watch condition. The values stored by different srluts in the scan chain are then shifted into the FPGA using the utility; No partial or complete recompilation of design is necessary. The scan chain, while it permits changing the watch condition and its value, does not allow changing the signals to be monitored on-line. To change the signals connected to the srluts, methods such as guided place and route[18], Xilinx FPGA editor[19] or JBits[20] can be used. Only the routing in the design has to changed, because the condition specification for the new signal can be calculated and a new stream of bits can be shifted into the srluts.

7 Design flow for the scan chain

The scan chain can be added either into the VHDL design or in the edif netlist. In this paper we have added it into the VHDL design. The srluts are instantiated as a component in the VHDL design flow, where they are assigned some initialization attributes. One or more instance of an srlut may be present in the design depending upon the number of signals to be monitored. In the VHDL code the design signals are directly connected to the input lines of an srlut, whereas the variables declared in a process are first stored in a signal which is then connected to the input of the srluts. The srlut inputs must be of type std_logic, thus the signals of type 'bit' or 'integers' have to be converted into std_logic. The pseudo code below illustrates the steps involved in the design flow.

1. *Add srluts into VHDL description and attach watch-point signals into the address lines of srluts.*
2. *Save the serial order information of different instance of srlut and corresponding signals (to be monitored) connected to them*
3. *Synthesize, place and route the design*
4. *To change the watch-point condition or watch value, input new condition and value along with the serial order information of srlut into the C program*
5. *Obtain the new bit sequence for the memory location of srluts in scan chain and download it to the FPGA*

6. *If a change in watch-point logic required then goto 4.*

8 Experiments and Results

To analyze the area overhead incurred by the scan chain, we used the Politecnico di Torino's ITC benchmarks (http://www.cad.polito.it/tools/#bench). These designs allowed us to insert scan-chain based watch-point logic both at VHDL and at the netlist level. Furthermore, these designs are of varying complexity, allowing us to analyze the watch-point logic insertion approach for a range of design sizes. The ITC benchmark suite consists of 22 different VHDL descriptions. We have used 14 of the 22 circuits in the ITC suite. Designs b01, b02, b03, b06, b09 and b13 are pure finite-state machines (FSMs). Design b14 is a complex VHDL description of a finite state machine. Designs b05, b07, b08 and b12 are examples of architectures combining FSMs data paths, and memory blocks. We have modified these benchmarks slightly, to accommodate the ports: interrupt, interrupt acknowledge, and input data. The interrupt and interrupt acknowledge ports are necessary to notify the host that the watch-point condition has occurred, the input data port is used to shift the user desired watch-condition into the srluts.

The target device in our experiments is Xilinx's XC2V250 chip. Design b14 did not fit into an XC2V250, and thus we have used an XC2V1500 for it. Table 2 shows the number of signals monitored in each of the benchmark circuits, the FPGA resource utilization by the original circuit and the resource utilization with the scan-chain based watch-point logic. Fig 4 shows the percentage increase in the slice count in scan-chain based implementation. The average area overhead of all of the benchmark circuits is 46% of the original area. During our experiments, we observed that the area overhead due to watch-point logic depends upon the following factors:

- Number of signals monitored
- Clock control or interrupt state machine
- State machine to shift data into srluts
- Shift mechanism (serial or parallel)
- Number of variables monitored
- Design congestion resulting in route through LUTs
- State machine encoding

The total number of srluts to connect N bits of watch-point signals is N/4. However, more srluts may be required, to connect output signals from all of the srluts together. For example, if N is 8 bits then three 4 input srluts are required, two for the watch-point signals and one for connecting the outputs of other two srluts. In general, there are $\log_4(N)$ levels of LUT logic needed, for values of N which are powers of 4, there are $N/4^i$, luts needed at each level, i. In Virtex-II FPGA each slice has two luts which can be configured as srluts. Thus, with the increase of every two srluts a slice is increased.

bench-	#	Original design			Design with scan chain		
		FF	LUT	Slice count	FF	LUT	Slice count
B01	8	15	17	11	18	28	16
B02	7	12	13	8	16	25	14
B03	18	38	66	42	43	96	59
B04	76	70	242	125	90	286	153
B05	67	26	259	144	50	366	203
B06	3	15	28	15	19	39	22
B07	50	52	127	72	68	179	105
B08	40	27	51	34	58	113	68
B09	27	37	69	35	44	114	61
B10	14	27	51	33	34	82	49
B11	30	37	218	120	50	254	141
B12	46	145	408	268	154	464	277
B13	42	67	69	55	70	132	81
B14	122	218	8,168	4092	261	8,280	4,160
Avg.	39	56	699	361	70	747	386

Table 2: Area overhead due to watch-point logic

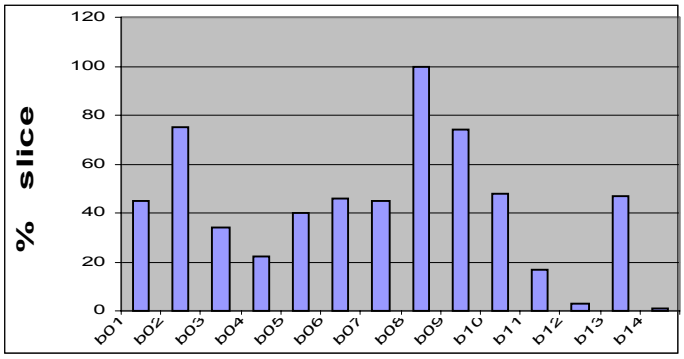

Fig 4: percentage of slice area overhead for watch-point logic

The clock control state machine has a fixed area overhead; it is a simple state machine with three states. The clock to the design is disconnected, whenever this state machine is in the interrupt state and if the interrupt is asserted due to watch-point logic. Most of the reconfigurable computing applications have such an interrupt state machine, which they use to assert an interrupt to the host when done computing. With this assumption, that the design has such state machine, we have calculated the area overhead without it. If there is any application which does not use an interrupt, the area overhead due to clock control machine has to be added to the overall area overhead. The area overhead of this clock control state machine is shown in table 3.

Clock control state machine	FF	LUT	Slice count
	4	6	4

Table 3: Area overhead due to clock control state machine

If the signals representing states of a finite state machine are to be monitored, the choice of state encoding method has a direct affect on the area overhead. One-hot encoding, in which setting a bit in the bit_vector represents the state increases the area overhead, as there are more signals to be monitored. In the ITC benchmarks circuits, FSM states are encoded as integers and the state signal is a bounded integer of size $\log_2(N_{states})$ bits. To monitor these signals, we converted them into std_logic_vector. To monitor the variables in a process, they were first stored in a signal, which is then connected to the srluts. We have observed that the signals kept to store the values of variables are synthesized as latches, thus, the number of variables monitored in the design increases the flip-flop utilization of the FPGA.

When the condition specification is serially supplied by the host, the state machine which shifts the data into srluts is a simple counter. When the condition specification is transferred in parallel to the FPGA, the state machine is more complex and has more area overhead. In both cases the area overhead increases as the number of srlut increases. The number of bits to be shifted equals $16*N_{srlut}$, the counter size required is $\log_2(16*N_{luts})$ for serial input of the condition spec. N_{srlut} is number of srluts in the design. Figure 5 shows the comparison of area for the original design without watch-point logic, to the area of the design with serial watch-point data input and the area of the design having parallel watch-point value input.

Routing congestion also may increase the area overhead when watch-point logic is added. Due to limited routing resource some of the neighboring LUTs of the monitored signals are used as route though LUTs and are left unused. For example, in the design b09 adding a signal to monitor a two-bit variable increases the flip-flop count by 2 and the LUT count by 4. Ideally, there should be an increase of two flip-flops with no change in the LUT count.

We have compared the area overhead of the scan-chain based watch-point logic to the area overhead of chipscope. Table 4 and 5 show the area overhead of chipscope and scan-chain watch-point logic respectively. It can be observed that scan-chain watch-point logic has significantly lower area overhead than chipscope for the same data width. This is in part due to the compact run-time modifiable comparison logic in our scheme which alleviates the need for a more general purpose comparator.

We have observed that in the scan-chain based technique, the srlut consumption increases more quickly than the flip-flop consumption when the watch-point width increases. This leaves many slices with unused flip-flops. These unused flip-flops can be used to latch the process variables and to implement edge-triggered watch-point conditions to catch events such as the rising edge or falling edge of a signal.

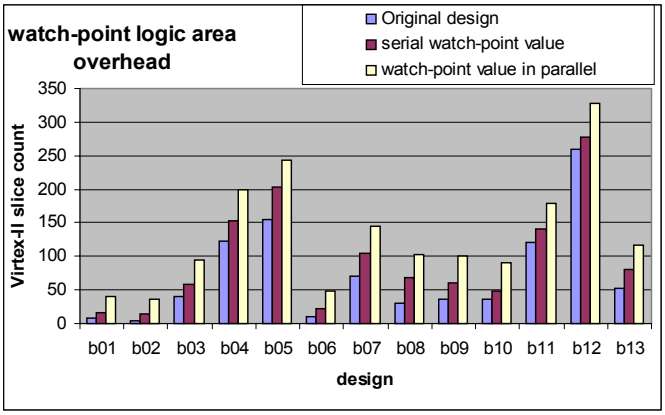

Fig 5: cost of original benchmark, benchmark with watch-point logic data coming serially and in parallel.

Watch-point data width	Flip-Flops	LUTs	Slices	Percentage of XC2V250 slice
2	125	143	72	4.6
4	128	151	84	5.4
8	134	167	84	5.4
16	146	199	100	6.5
32	173	265	133	8.6
64	222	395	198	12.8

Table 4: Chipscope area overhead

Watch-point data width	Flip-Flops	LUTs	Slices	Percentage of XC2V250 slice
2	7	10	7	0.45
4	9	18	12	0.78
8	11	24	15	0.97
16	12	31	19	1.27
32	13	39	23	1.49
64	14	56	32	2.08

Table 5: Scan chain area overhead

8.1 Trace buffer

Debugging of certain applications can be facilitated if in addition to watch-point capability, a history of the monitored signal's value is available to the user. This helps a user in determining how a signal has changed during the clock cycles prior to the occurrence of the condition. To maintain the history of a signal's values a trace buffer can be added into the design in conjunction with the watch-point logic.

To implement the trace buffer we use block RAMs provided in the Xilinx Virtex-II series of FPGAs. Block RAMs are high-speed SRAM modules available inside an FPGA. They can be configured in different width and depth combination, for example in the Virtex-II series of FPGA from Xilinx the following combinations are available: 1x16384, 2x8192, 4x4096, 9x2048, 18x1024, and 36x512. Connecting multiple Block RAMs together can create wider and/or deeper Block

RAMs. A FIFO can be constructed by connecting a counter at the address lines of the Block RAMs. This FIFO is used as a trace buffer, signals to be traced are connected at the data lines of the Block RAM and the address counter is incremented on every clock cycle. The counter used in the FIFO should be able to address all the locations in the Block RAM, thus the counter size depends on the depth of a FIFO. Table 6 shows the area overhead of implementing a FIFO using a single Block RAM with varying width and depth. It is observed that it is the depth of FIFO which causes change in the area overhead (LUT, FF and slices), increasing the width of a traced signal merely increases the Block RAM count. For example, if 512 traces of a signal are required, signals with 36, 72 and 144 bits will use 1, 2 and 4 Block RAM respectively, but will have the same slice count for the FIFO control logic. This is due to the fact that the same counter and control logic can be used to address multiple Block RAMs.

Trace buffer depth	Flip-Flops	LUTs	Slices
256	31	34	22
512	31	34	22
1024	34	35	22
2048	38	38	26
4096	41	40	27
8192	44	43	30
16384	47	45	30

Table 6: Area overhead of trace buffer control logic

9 Conclusions

In this paper we have proposed a structured, fast and area efficient technique to implement hardware watch-points. This technique enables run-time modification of watch-point condition and its value without any design recompilation or reconfiguration. The watch-points implemented using this technique provide software style interactive debugging and make the debugging process many times faster than hardware simulation. The debugging speedup is the result of running and debugging the design directly on the target hardware platform. The limitation of this technique is that the signals, which are monitored, cannot be changed at run-time, some degree of design recompilation is necessary to achieve that. Future work can be done to automate the process of inserting multiple instances of srluts into the vhdl design.

References:

[1] B.L. Hutchings et. al. *A CAD suite for high performance FPGA design*, proceedings of IEEE symposium on Field-Programmable Custom Computing Machines, April 1999.

[2] B.L. Hutchings et.al. *Unifying Simulation and Execution in a Design enivornment for FPGA Systems* IEEE trans on VLSI, Feb'00

[3] K. A. Tomko and A. Tiwari. *Hardware/Software Co-debugging for Reconfigurable Computing* IEEE International High Level Design Validation and Test workshop, Oakland CA, Nov. 2000

[4] Virtex FPGA series configuration and readback. Application Note XAPP138, Xilinx San Jose CA, October 2000

[5] Lucent Technologies, *ORCA Series 4 FPGAs*, Dec 2000

[6]T.Wheeler et. al. *Using design-level scan to improve FPGA design observability and controllability for functional verification* FPL'01

[7] Paul Graham et. al. *Instrumenting Bitstreams for Debugging FPGA Circuits,* proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, April 2001

[8] Xilinx, San Jose CA. *ChipScope software and ILA Cores User Manual*, v. 1.1. June 2000

[9]Altera, San Jose CA. *SignalTap Embedded Logic Analyzer Megafunction*, April 2001 ver.2.0

[10] Triscend Inc. E5 Configurable System-on-Chip Platform data sheet, July 2001 (ver. 1.06)

[11] SIDSA Inc, SF CA, FIPSOC™ Mixed Signal System-on-Chip.

[12]FIPSOC user manual chapter 7, SIDSA Inc.

[13] P. Graham, *Logical Hardware Debuggers for FPGA-Based Systems*, PhD Thesis, Brigham Young University, Electrical and Computer Engineering Department, Dec. 2001

[14] K. A. Tomko, A. Tiwari, *Design Techniques to Implement Reconfigurable Hardware Watch-Points for Hardware/Software Co-Debugging*, Proceeding of the Conference on Engineering of Reconfigurable Systems and Algorithms, June 2001.

[15] Altera corp, Stratix Programmable Logic Device Family Data Sheet, version 2.0 April 2002

[16] Xilinx Inc, Virtex-II Platform FPGA Handbook, ver. 1.3 Dec' 01

[17] M. Abramovici, M.A. Breuer, A.D. Friedman. *Digital Systems testing and testable design pp. 358* IEEE press 1990

[18] Using Xilinx and Synplify for Incremental Designs (ECO), Xilinx application note XAPP164, Xilinx San Jose, CA 1994

[19]Xilinx Inc, Xilinx 4 Software Manuals.

[20] S. A. Guccione, D. Levi, and P. Sundararajan, *JBits: A Java-based interface for reconfigurable computing*, Proceedings of the 2nd Annual conference on Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD), September 1999.