Digital Design for DSP & Communication

EE-278

Mini Project - V

LMS Adaptive Filter – Hardware Accelerator

Under supervision of

Dr. Charles Chang Choo

By

Dhatri Patel

SJSU ID: 010692526

Software Implementation of Adder

```
#include "sys/alt_stdio.h"
#include "system.h"
#include "alt_types.h"
#include "stdio.h"
#include "io.h"
#include "altera_avalon_performance_counter.h"
#include "fircoeffB.h"
#include "inputB.h"
#define PC_ADDR 0x5200
#define HW_ADDER_INTERFACE_BASE 0x5000
int main()
{
	signed long int result=0;
	while(1){
		PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
		PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
		PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE,1);
		int i;
		for(i=0;i<N;i++){

			result += FirCoeff[i]+InSamples[i];
		}
		printf("result::%d\n",result);
		PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
		PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
		long time;
		time=perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,1);
		printf("cycles taken:%d",time);
		break;

	}
	return(0);


}
```

Result:

From below figure, we observe that it takes 27900 cycles to compute addition of 200 input samples.

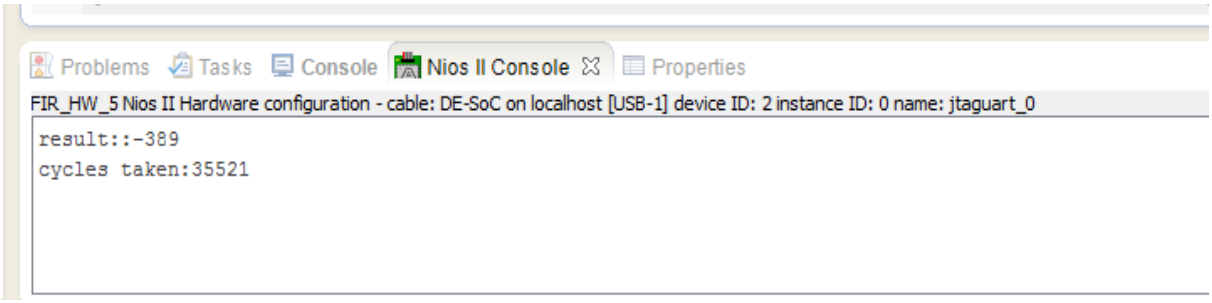

figure 1

Software Implementation of LMS Adaptive Filter

```
#include "stdio.h"
#include "stdlib.h"
#include "system.h"
#include <io.h>
#include "alt_types.h"
#include "altera_avalon_performance_counter.h"
#include "fircoeffB.h"
#include "inputB.h"

int main(void){
        signed short int OutSamples[NOUT] ={0};
                signed short int SampleBuf[N] ={0};
                signed short int FirCoeff[N] ={0};
                signed short int desired_d =0;
                short i,j,k,m=0;
                signed long int yout=0;
                signed int error[N]={0};

        PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
        PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
        PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
yout=0;
```

```
        for (i=0;i<NOUT;i++){
                        SampleBuf[0]=InSamples[i];



                        for(j=0;j<N;j++){
                                yout += FirCoeff[j]*SampleBuf[j];
                                }

                        OutSamples[i] = yout>>20;
                        error[i]= desired[i]-OutSamples[i];


                        for(m=0;m<NOUT;m++){
                                FirCoeff[m]=(22*error[m]*SampleBuf[m]);

                        }
                        for(k=(N-1);k>0;k--){
                                SampleBuf[k]=SampleBuf[k-1];
                        }

                        yout=0;
                }
                PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
                PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
                long time;
                time=perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,1);
                printf("\n SO it takes %1d cycles\n",time);
                for(i=0;i<NOUT;i++){
                printf("desired =%d %d error=%d\n",desired[i],OutSamples[i],error[i]);

                }


        return(0);
}
```

Result:

The above code was simulated on NIOS II software with 200 input samples. We observed that it takes 50042803 cycles to compute the result. From Figure 2, we observe that error value has become constant near about zero. There is some problem with software as I tried to copy the results from there, software crashes as shown in figure 3 and so I have to restart the process again.

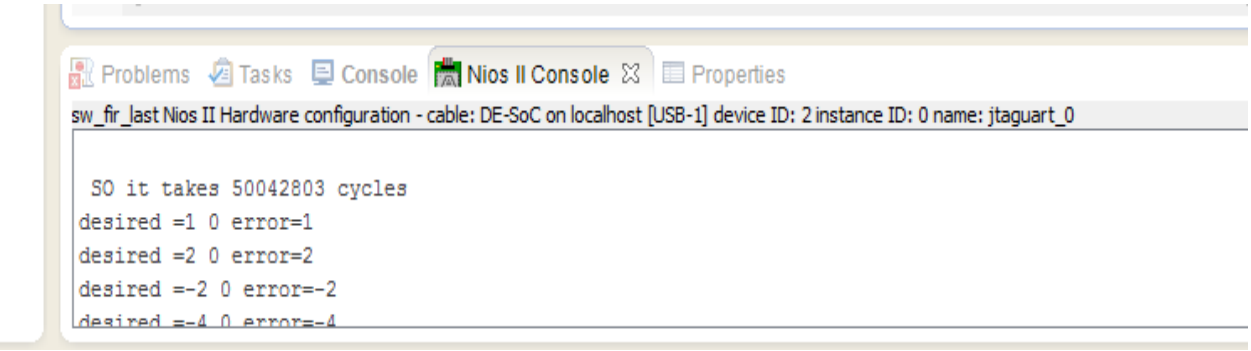

figure 2

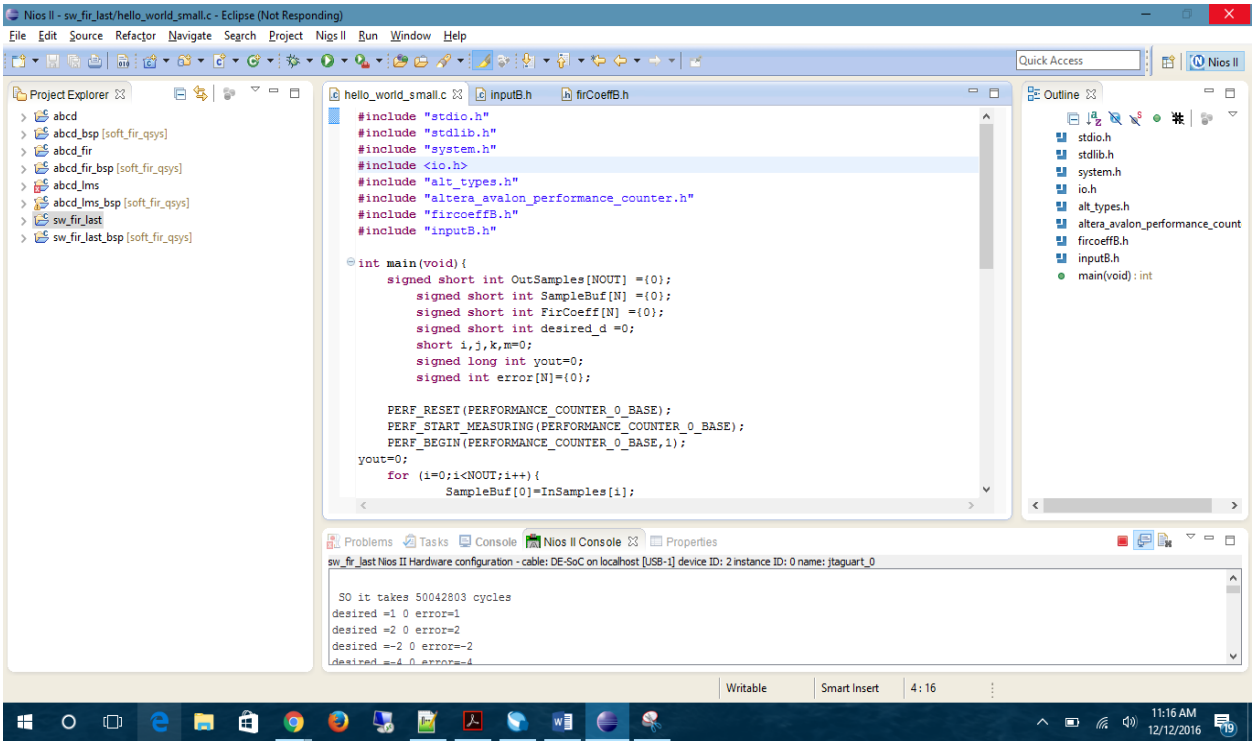

figure 3

Hardware Acceleration for Adder

```
#include "sys/alt_stdio.h"
#include "system.h"
#include "alt_types.h"
#include "stdio.h"
#include "io.h"
#include "altera_avalon_performance_counter.h"
#include "inputA.h"
#include "inputB.h"
#define PC_ADDR 0x5200
#define HW_ADDER_INTERFACE_BASE 0x5000
int main()
{
	signed long int result=0;
	while(1){
		PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
		PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
		PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE,1);
		int i;
		for(i=0;i<N;i++){
			IOWR(HW_ADDER_INTERFACE_BASE,1, inputA[N-i]);
			IOWR(HW_ADDER_INTERFACE_BASE,2,InSamples[i])
		}
		//result=IORD(HW_ADDER_INTERFACE_BASE,0);
		printf("result::%d\n",result);
		PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
		PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
		long time;
		time=perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,1);
		printf("cycles taken:%d",time);
		break;

	}
	return(0);


}
```

Result:

From results, we observe that number of cycles to compute addition of 200 input samples is 27900. The cycles in respect software implementation took 35521 cycles, thus hardware implementation takes less number of cycles. Less number of cycles means faster system and less power consumption.

figure 4

Hardware Acceleration for LMS Adaptive filter

In QSYS software I have created a filter component which will be used for hardware acceleration. In figure 5 we can see that component. In figure 6 the system connection in QSYS software is shown. The system comprises of a NIOS II processor, performance counter, JTAG UART, On-chip memory and user created filter component.

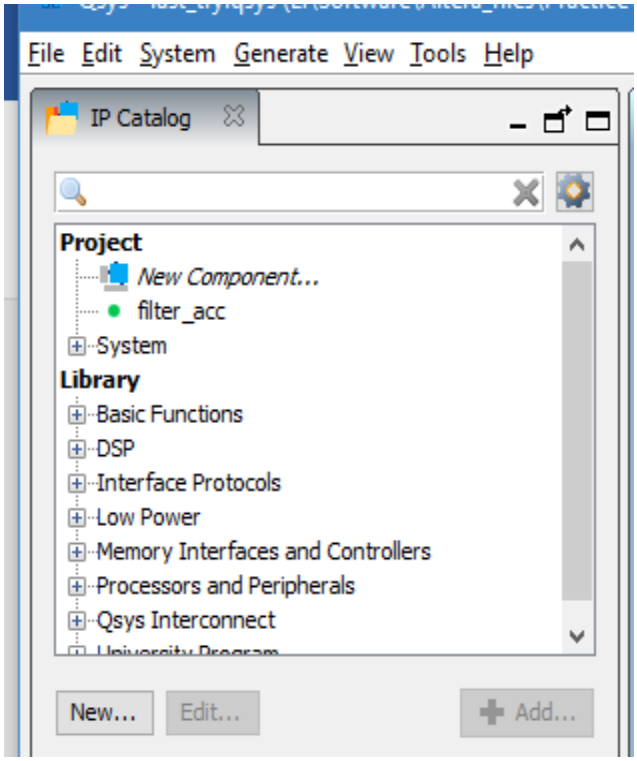

figure 5

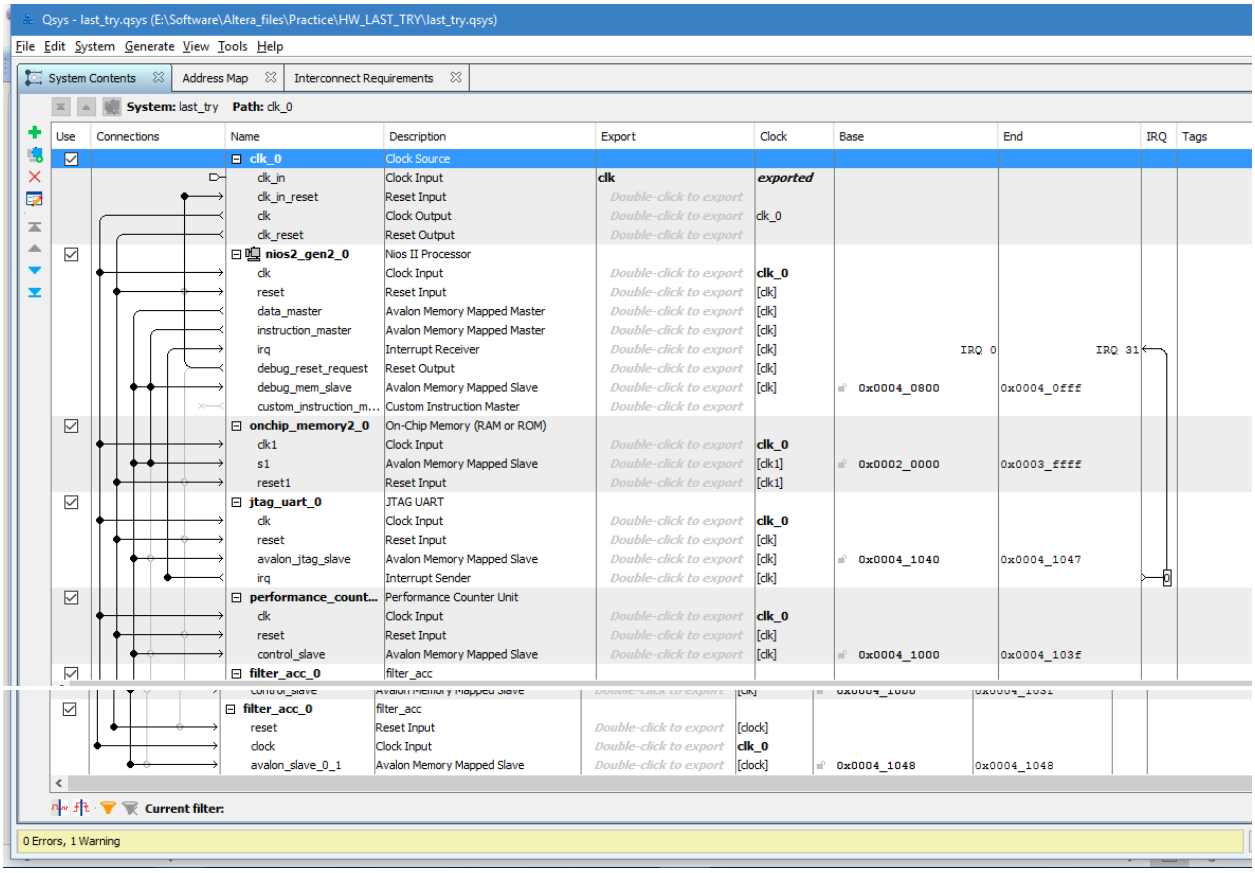

figure 6

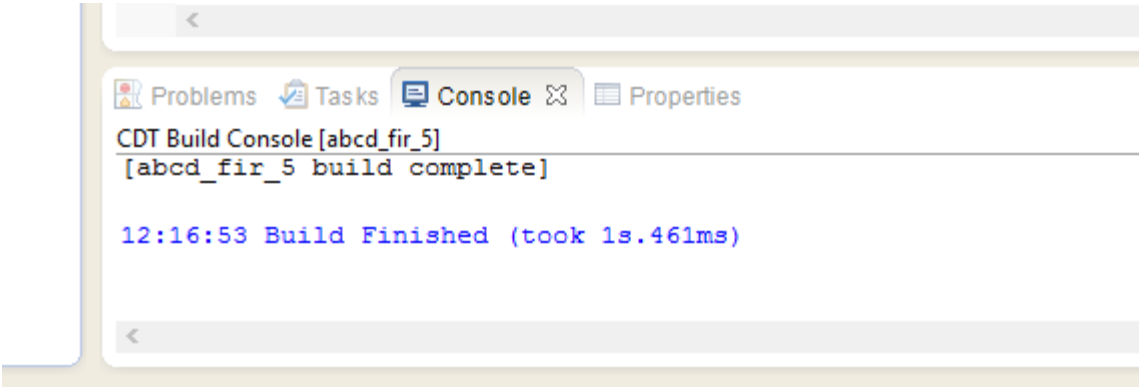

figure 7

In figure 7, we observe that build process for hardware accelerator is also successful, but when we run it on NIOS II hardware then we came across an error message as shown in figure 8. I tried to search about it on internet and found out that we must refresh the connections. I tried that but it still did not work.

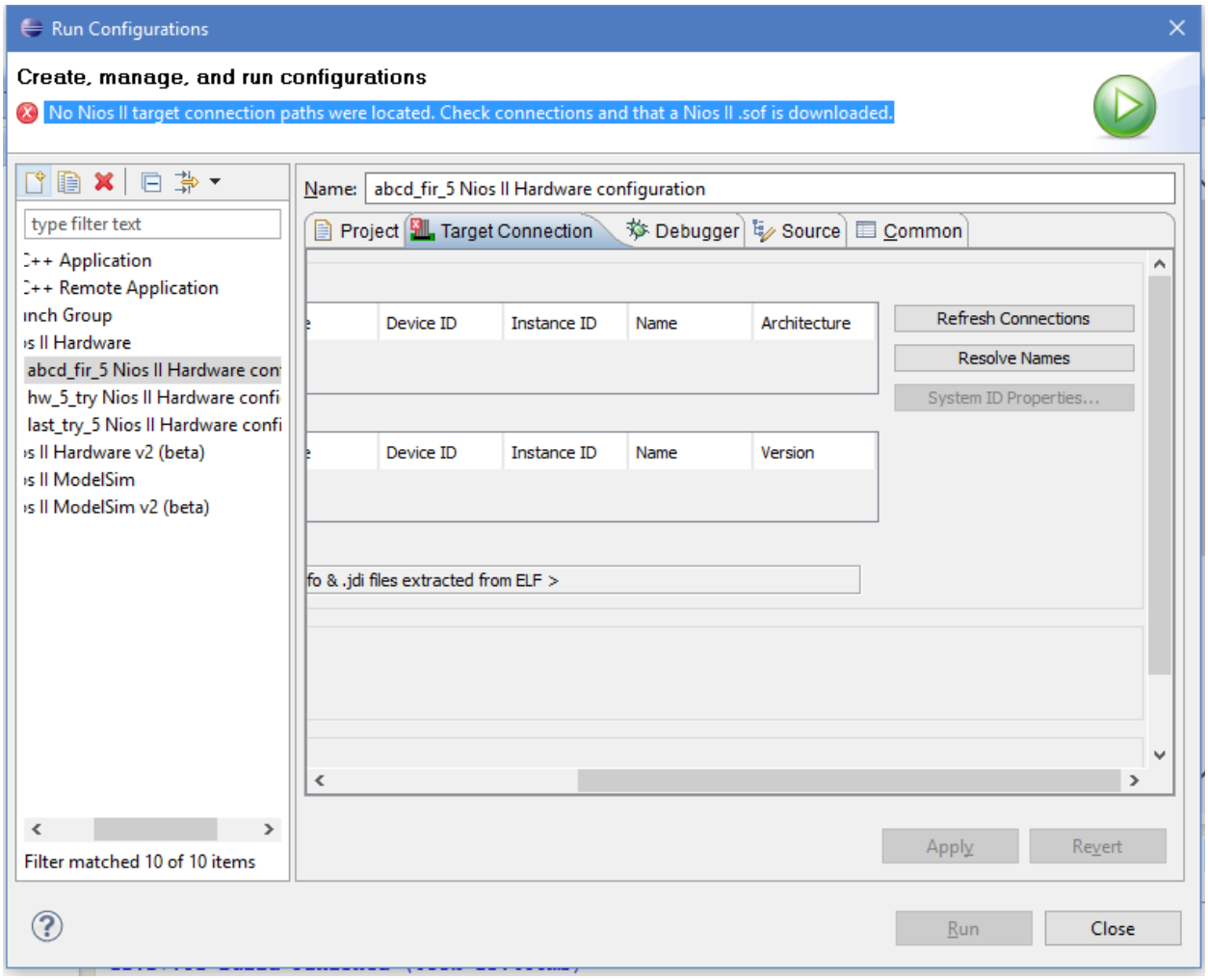
Run Configurations
Create, manage, and run configurations
No Nios II target connection paths were located. Check connections and that a Nios II .sof is downloaded.
type filter text
C++ Application
C++ Remote Application
inch Group
s II Hardware
abcd_fir_5 Nios II Hardware con
hw_5_try Nios II Hardware confi
last_try_5 Nios II Hardware confi
s II Hardware v2 (beta)
s II ModelSim
s II ModelSim v2 (beta)
Filter matched 10 of 10 items
Name: abcd_fir_5 Nios II Hardware configuration
Project
Target Connection
Debugger
Source
Common
Device ID
Instance ID
Name
Architecture
Refresh Connections
Resolve Names
System ID Properties...
Device ID
Instance ID
Name
Version
fo & .jdi files extracted from ELF >
Apply
Revert
Run
Close

figure 8

Conclusion

In a nutshell, this project is all about implementation of LMS adaptive filter – Hardware accelerator. To implement this, we started by implementing first stage that is adder. By implementation of adder on software and hardware part it can be observed that number of cycles to perform addition is reduced in hardware implementation as compared software.

The second stage is to implement LMS adaptive filter in software. In software implementation of LMS adaptive filter we observe that number of cycles taken are 50048203. If implementation of hardware was successful, then fair comparison of hardware accelerator LMS adaptive filter can be done. Hardware accelerator are quite useful when a designer wants to compute results from a block in less number of cycles. Also great amount of power is saved as number of cycles reduces by a great scale.