CS701 GRP Phase 2 Report

Group 7 - Beck Busch, Frank Shen, Yogesh Dangwal

1. Introduction

This project centered on the design and development of a basic processor and a suite of application-specific processors (ASPs) that can be configured and operated by the processor as part of a cascading sequence. This will require an understanding of the TDMA NoC system to connect these components together through an efficient communication protocol. The processor we are designing is based on ReCOP, a relatively simple processor design that integrates well with other processors and ASPs. The instruction set for ReCOP is similar to MIPS, making it intuitive to understand and develop. Unique extensions to the ISA allow the processor to interact with components in the environment and be debugged easily when running in parallel with a co-processor.

2. NOC Structure

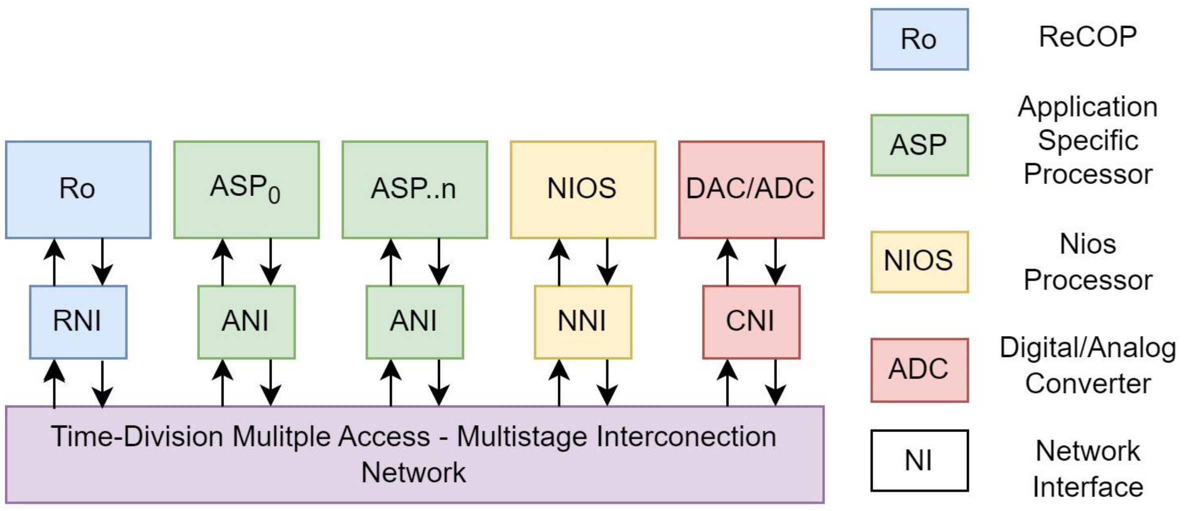
Heterogeneous Multiprocessor System-on-Chip (HMPSOC) is a type of System-on-Chip (SoC) that combines a variety of processors to provide performance, power efficiency, and cost balance. These general-purpose or specialized processors demand various design approaches and tools, such as hardware-software co-design and system-level design tools. The communication and interconnect infrastructure, such as buses, crossbars, or Network-on-Chip (NoC), is critical to the design of HMPSOCs because it allows different types of processors and components to interact and coordinate operations.

2.1. NOC Architecture

TDMA-MIN NoC is a type of network that is designed to manage tasks and communications in a very precise and predictable way. This is great because it ensures that important communication happens exactly when it needs to.

To accomplish this, TDMA-MIN NoC combines two methods, Time Division Multiple Access (TDMA) and Minimal Indirection (MIN), which is a kind of connection similar to a Banyan tree's branching structure. TDMA-MIN NoC allows each source (let's say,DAC ASP and ADC ASP) to send a packet (a piece of data) to any destination (node) within a set time period, known as a TDMA round. During each time slot of the TDMA round, it can also handle multiple connections between other sources and destinations.Also it has lower implementation cost than a traditional MIN or mesh network.

Figure 1: TDMA-MIN Diagram



3. ReCOP Processor

The ReCOP processor is a coprocessor to the NIOS processor that is specifically designed to configure the multiple application specific processors connected on the nodes of the TDMA network. The purpose of ReCOP is to be a reactive processor that responds to stimulus from the environment to configure the ASPs with low latency.

3.1. Control Unit

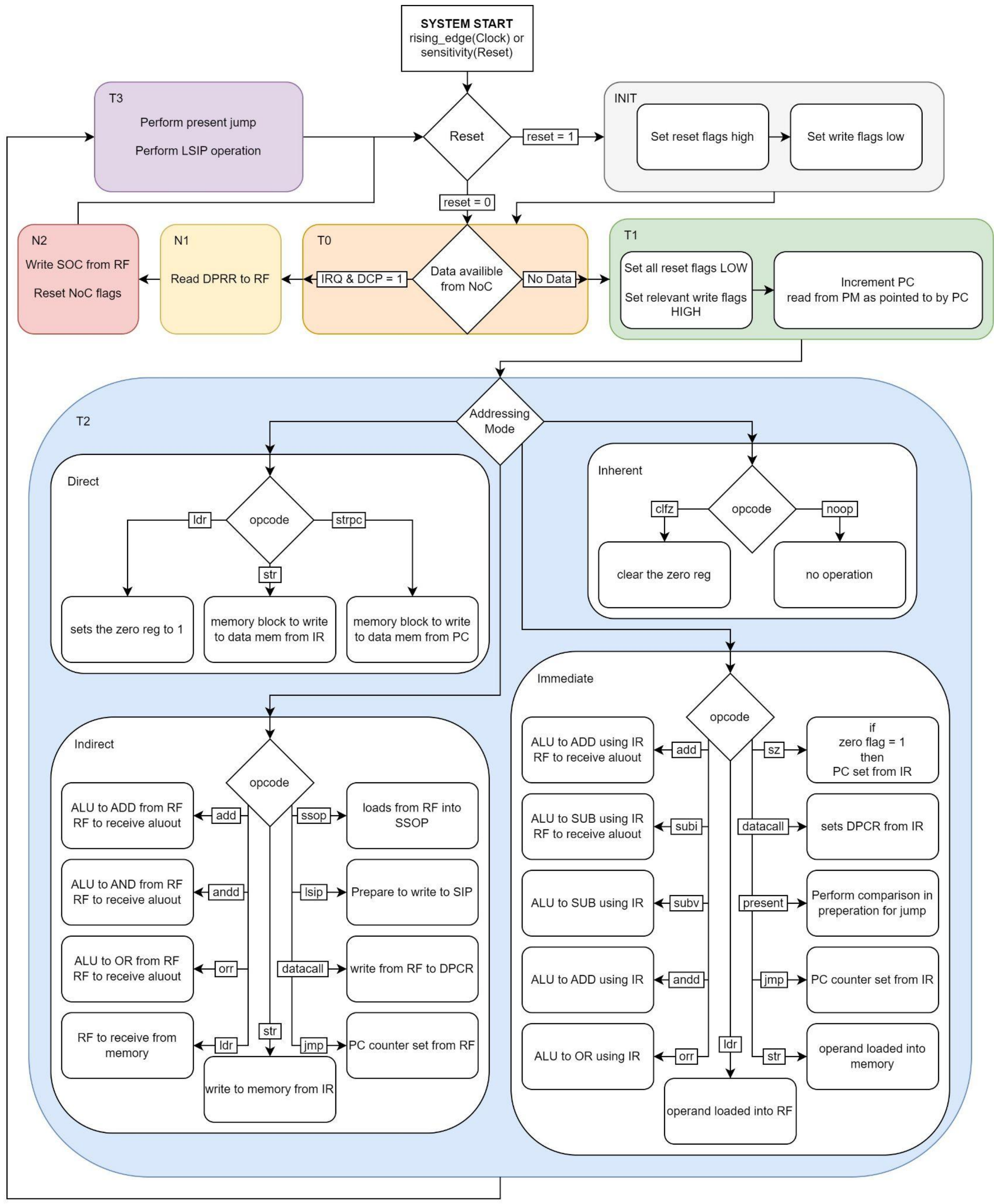
3.1.1. Control Unit Architecture

The control unit has two blocks; Pulse Distributor and Opcode decoder. Pulse distributor cycles through a set of different states in a predetermined order. Each of these states lasts for a duration of one system clock cycle. in each cycle, or "tick" of this system clock, the pulse distributor moves from one state to the next. This shift from one state to another happens at the rising edge of the clock cycle, which is the transition from a low to a high signal in the cycle. The majority of actions occur on the falling edge, except for the control unit reset, which is asynchronous, and register writes, which occur on the falling edge. This design decision was made so that we can read and write to a single register in the same cycle instead of across two different cycles.

3.1.2. Control Unit FSM

The ReCOP Control Unit uses a Moore Type state machine. There are four different states in our FSM; Initialisation, T1, T2, and T3. The ReCOP system executes instructions using a series of "micro-operations." A micro-operation is a fundamental operation performed by computer hardware, such as data transfer between registers. The ReCOP system completes each instruction in three phases. Each of these steps is represented by a machine cycle that includes obtaining an instruction, decoding it, executing it, and storing the result. T1, T3, and T3 are the three machine cycles involved in the execution of a typical ReCOP instruction. Each cycle indicates a new step in the execution of instructions. The control stages are outlined as follows.
T0: Test DPC cycle
In this state, we check DPC and IRQ. We then transition to the N1 state if both flags are true, otherwise we transition to the T1 state.
T1: Instruction fetch cycle
This state is the first phase of executing an instruction in the ReCOP system.
The 32 bits of the instruction are fetched from the location pointed by the program counter in program memory and are loaded into the Instruction Register (IR). This is where the fetched instruction is temporarily stored while it is being executed. The Program Counter (PC) is then incremented by one, so the PC will point to the location of the next instruction in the sequence, which will be ready to be fetched in the next cycle. This instruction will be executed in the next cycle unless something happens to change the flow of the program (like a jump instruction, which would cause the PC to point to a different location). The T1 phase remains the same for all instructions independent of the addressing mode.
T2: **Instruction decode cycle**
We perform the instruction based on the values stored in the instruction register. Depending upon the addressing mode and opcode value, we configure various datapath signals to retrieve the values and perform the required operations.‘Present’ and ‘Lsip’ instruction is not executed in this state, we only compare the ‘rz’ register value with zero.
T3: Signal reset cycle
In T3, we reset various datapath signals to their default values and perform the second half of the present operation. When performing a present operation, we use the result of the comparison performed in T2 to decide if we jump the program counter or continue as usual.
N1: Write dprr
In this state, we write the contents of the DPRR register to the register file before moving to N2. This state is crucial for the ReCOP processor to communicate with the NoC and the ASPs.
N2: Reset
In this state, we reset DPC and IRQ flag. Before resetting, we write register 10 value to the ’ Sop’ register, then return to T0.

Figure 2: Control Unit Diagram



3.2. Datapath

The datapath consists of 5 functional blocks that organize data and perform operations. This is the core of the processor's functionality, moving and operating on data to achieve the desired result.

Program Counter (PC)

The program counter consists of a 4x1 multiplexer and a single 32-bit register. The multiplexer selects between memory, the register file, the instruction register, and an increment of the pc register. This value is then stored in the pc register and is used to direct the reading of program memory.

Instruction Register (IR)

This block is implemented as a 32-bit register that is loaded with values from data memory in a single step. This allows us to easily access the current program step and access individual parts as needed.

Register File (RF)

The register file consists of 16 16-bit registers and several multiplexers to control data flow. An 8x1 multiplexer selects an input to the register block, while a 2x1 multiplexer selects the location to store the data. Another 2x1 multiplexer selects the main output of the RF alongside several other outputs that relay the contents of specific registers.

Arithmetic Logic Unit (ALU)

The ALU block consists of two multiplexers that receive inputs and an ALU that performs several basic arithmetic operations. These operations are addition, subtraction, logical or, and logical and. As well as outputting the result of the arithmetic operation, the ALU outputs a flag whenever the result of the operation is zero. This can be used for more efficient control.

Memory Interface Block

The memory interface block outputs two values to select the address and contents of data to be stored in memory. We are able to select the data from the instruction register operand, instruction register rx, or dprr. The address can be selected from the program counter output or any value from the instruction register.

Figure 3: ReCOP Datapath

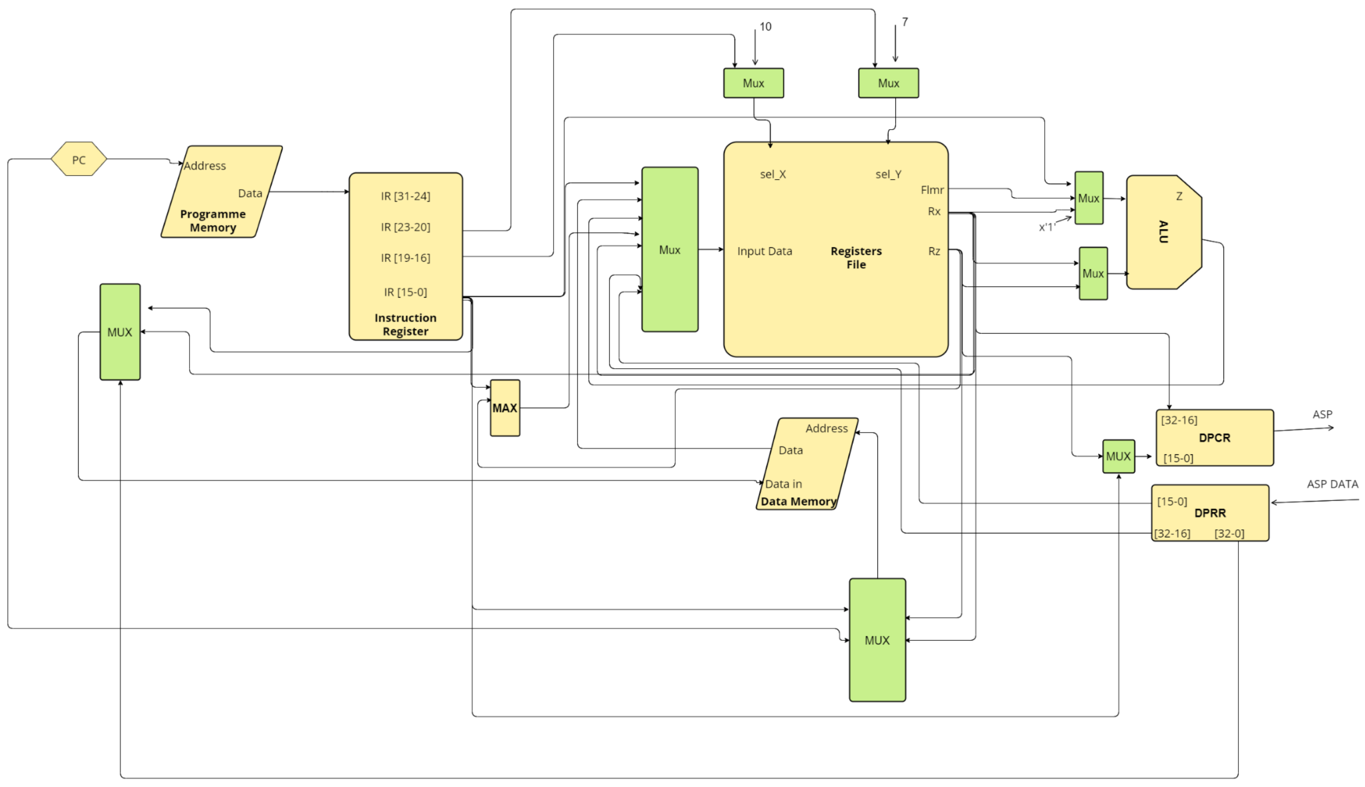
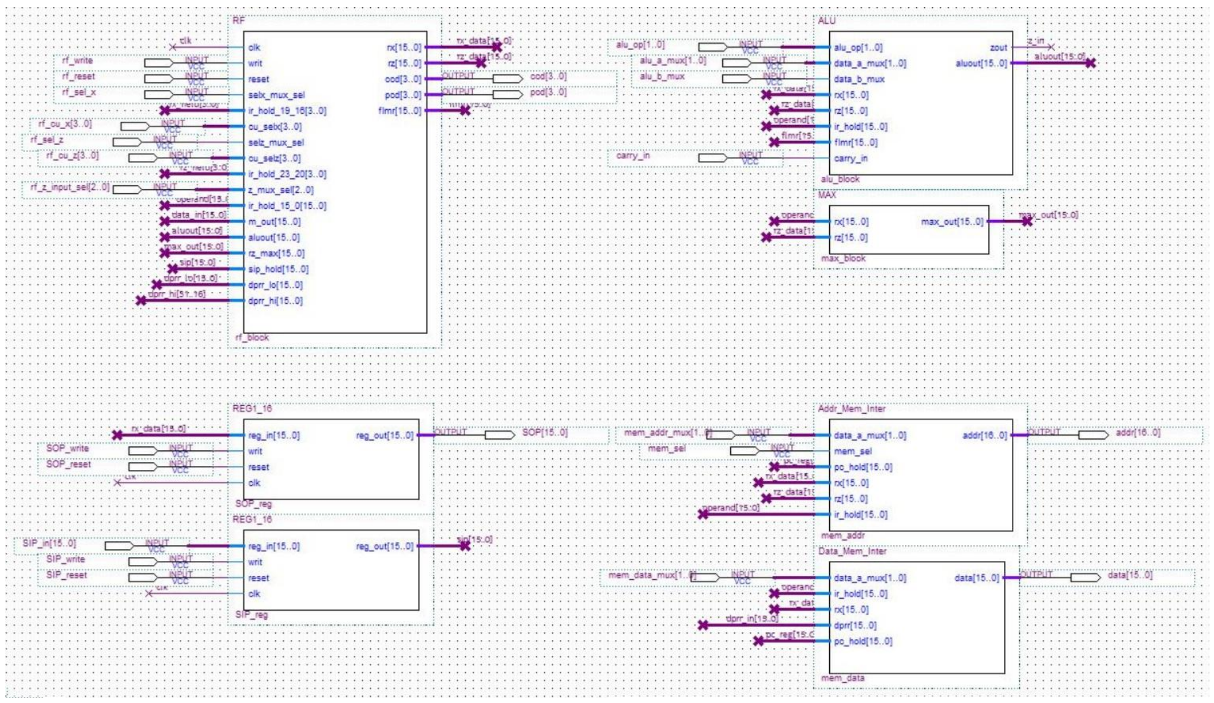


Figure 4: Excerpt of Datapath Quartus Implementation



3.3. Program Code Format

The processor reads instructions from program memory in a format that allows the parts of the instructions to be easily separated and used. All parts are stored together in 32-bit blocks consisting of a 16-bit long instruction word and a 16-bit operand. Our processor design always retrieves 32 bits, including the operand, regardless of whether an operand is required. In this case, we expect the operant to be blank values in the program memory. Altogether, the 32-bit blocks of instruction contain 5 distinct values;

3.3.1. Addressing Mode

The initial 2 bits of an instruction describe how values needed for the instruction are to be retrieved.
Inherent addressing mode instructions do not require unique values, as these usually set flags or alter parts of the datapath. Their effect is the same each time and therefore does not require any unique operation values.
Immediate instructions contain all the relevant information needed to perform the instruction within the 32-bit block. The destination address, source address, and an integer value are provided in rz, rx, and the operand, negating the need to locate more than one value from the datapath.
Direct instructions interact directly with memory, providing only a single register and a location in memory for which to carry out the instruction. Notable examples are store and load, which read and write values between a location in memory and a register in the register file.
Indirect instructions perform an operation between two different registers from the register file. No other values are provided, just the indexes of the two registers.

3.3.2. OpCode

The opcode is a 6-bit value that selects which instruction will be performed by the data path. These values are specified by the instruction set architecture and hard-coded into the control unit, with different opcodes requiring different configurations of the datapath.

3.3.3. Rz & Rx

Rz and Rx provide the indexes of two registers in the register file. Rz is commonly the destination where the result of the operation will be saved, while Rx is the source of one of, if not the only, value used in the operation.

3.3.4. Operand

The operand is a 16-bit integer that is used in some instructions, providing a way to introduce new arithmetic values into the processor. While Rz and Rx refer to dynamic values stored in ram, the operand is a static value stored in the program memory. Along with the opcode, the operand is responsible for most of the content that we would think of as actually running code on the processor, as opposed to maintaining the datapath and managing values.

3.4. Instruction Set Architecture

The ISA for ReCOP consists of several operations that can be performed with the processor to carry out basic programmatic functionality and communicate with other processors in its environment. These are the various opcodes used by the control unit. The instructions that we have implemented in our processor can be found below in Table 1.

Table 1: ReCOP ISA

Instruction	**Register Transfer**	**Addressing Mode**
AND Rz Rx #Operand	Rz <- Rx AND Operand	Immediate
AND Rz Rz Rx	Rz <- Rz AND Rx	Register
OR Rz Rx #Operand	Rz <- Rx OR Operand	Immediate
OR Rz Rz Rx	Rz <- Rz OR Rx	Register
ADD Rz Rz #Operand	Rz <- Rx + Operand	Immediate
ADD Rz Rz Rx	Rz <- Rz + Rx	Register
SUBV Rz Rx #Operand	Rz <- Rx - Operand	Immediate
SUB Rz #Operand	Rz - Operand	Immediate
LDR Rz #Operand	Rz <- Operand	Immediate
LDR Rz Rx	Rz <- M[Rx]	Register
LDR Rz $Operand	Rz <- M[Operand]	Direct
STR Rz #Operand	M[Rz] <- Operand	Immediate
STR Rz Rx	M[Rz] <- Rx	Register
STR Rx $Operand	M[Operand] <- Rx	Direct
JMP #Operand	PC <- Operand	Immediate
JMP Rx	PC <- Rx	Register
PRESENT Rz #Operand	if Rz(15..0)=0x0000 then PC<-Operand else NEXT	Immediate
DATACALL Rx	DPCR <- Rx & R7	Register
DATACALL Rx #Operand	DPCR <- Rx & Operand	Immediate
SZ Operand	if Z=1 then PC <- Operand else NEXT	Immediate
CLFZ	Z <- 0	Inherent
LSIP Rz	Rz <- SIP	Register
SSOP Rx	SOP <- Rx	Register
NOOP	No Operation	Inherent
MAX Rz #Operand	Rz <- MAX{ Rz, #Operand}	Immediate
STRPC $Operand	M[Operand] <- PC	Direct

4. Application-Specific Processor Design

Application specific processors are small scale processor blocks that can preform simple tasks. They are particularly powerful when used in parrelel with other ASPs, as chaining several together can create an efficient data processing pipeline.

4.1. ASP Configuration

ASPs configure themselves using the top 16 bits of the TDMA Data input. When the header of the data matches the configuration header for the particular asp, the asp will use the rest of the top 16 bits to set configuration values. These include the destination address where the ASP will direct its output and a mode switch to alter the functionality of the ASP. Various ASPs can have their own unique configuration values, such as channel selection for an audio ASP.

4.2. Rolling Average ASP

One of the ASPs in our sequence calculates the rolling average of the last n data points it receives. It does this by storing data in a shifting register and then calculating the summation of these values. The summation is then divided by n to provide the average. The number of data points it stores for the calculation is based on a value that can be set during configuration.

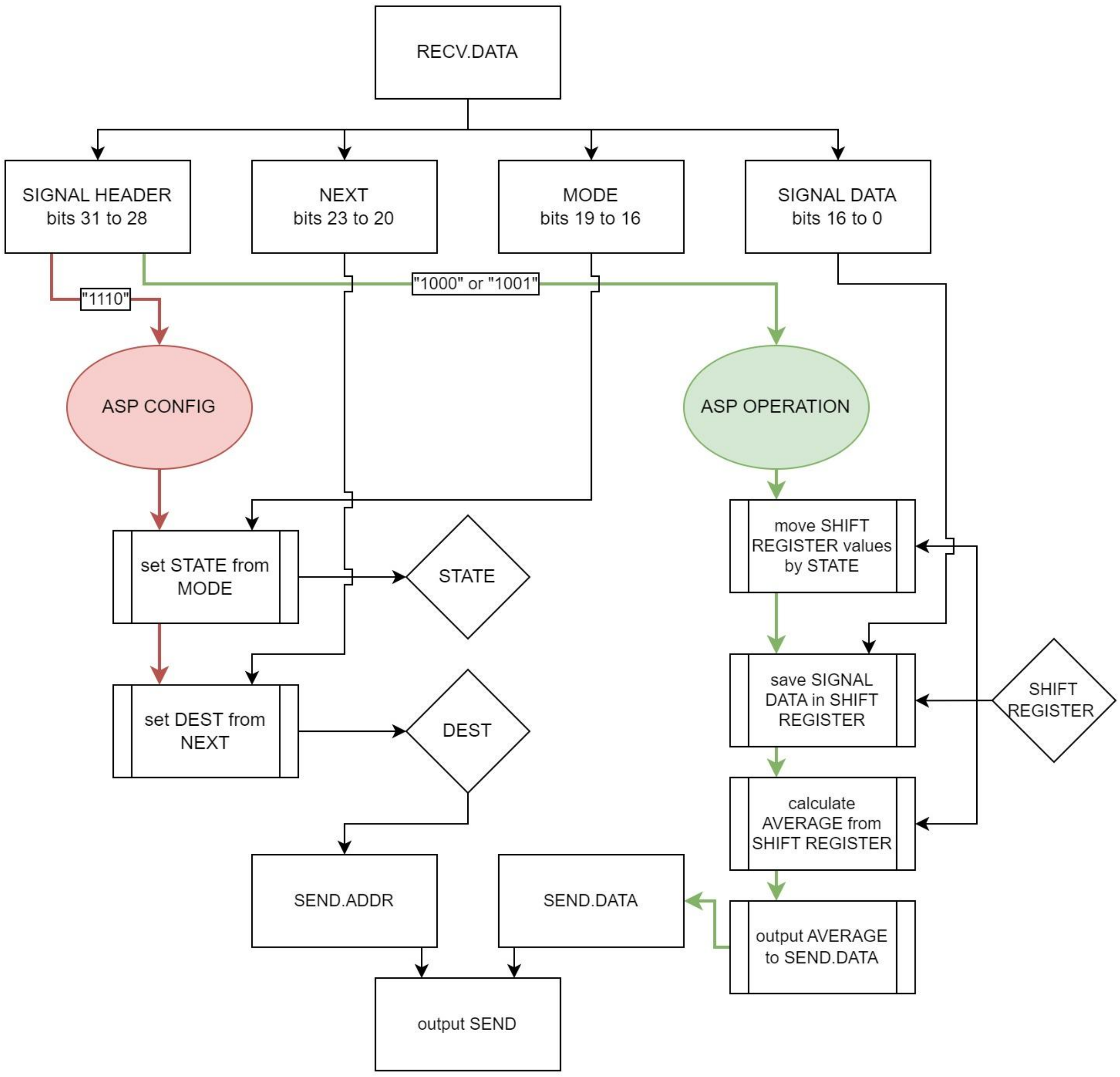
4.2.1. Average ASP Operation

The ASP can currently operate with two signal types, audio signal and data signal. When the data header is "1000", the ASP will operate the shifting registers and return an average value. The operation is as follows:

1. Reset the value of the summation register.
2. Switch based on the window size, controlling the style of operation.
3. Move the value in the required registers to the next register. The required registers are reg n-1 to reg 1, where n is the window size.
4. Store the new data point in register 1.
5. Sum the values of registers n to 1.
6. Divide the summation by the window size, then return this value.

This behavior is displayed in the program flow diagram below in Figure 3.

Figure 5: Rolling Average ASP Program Flow



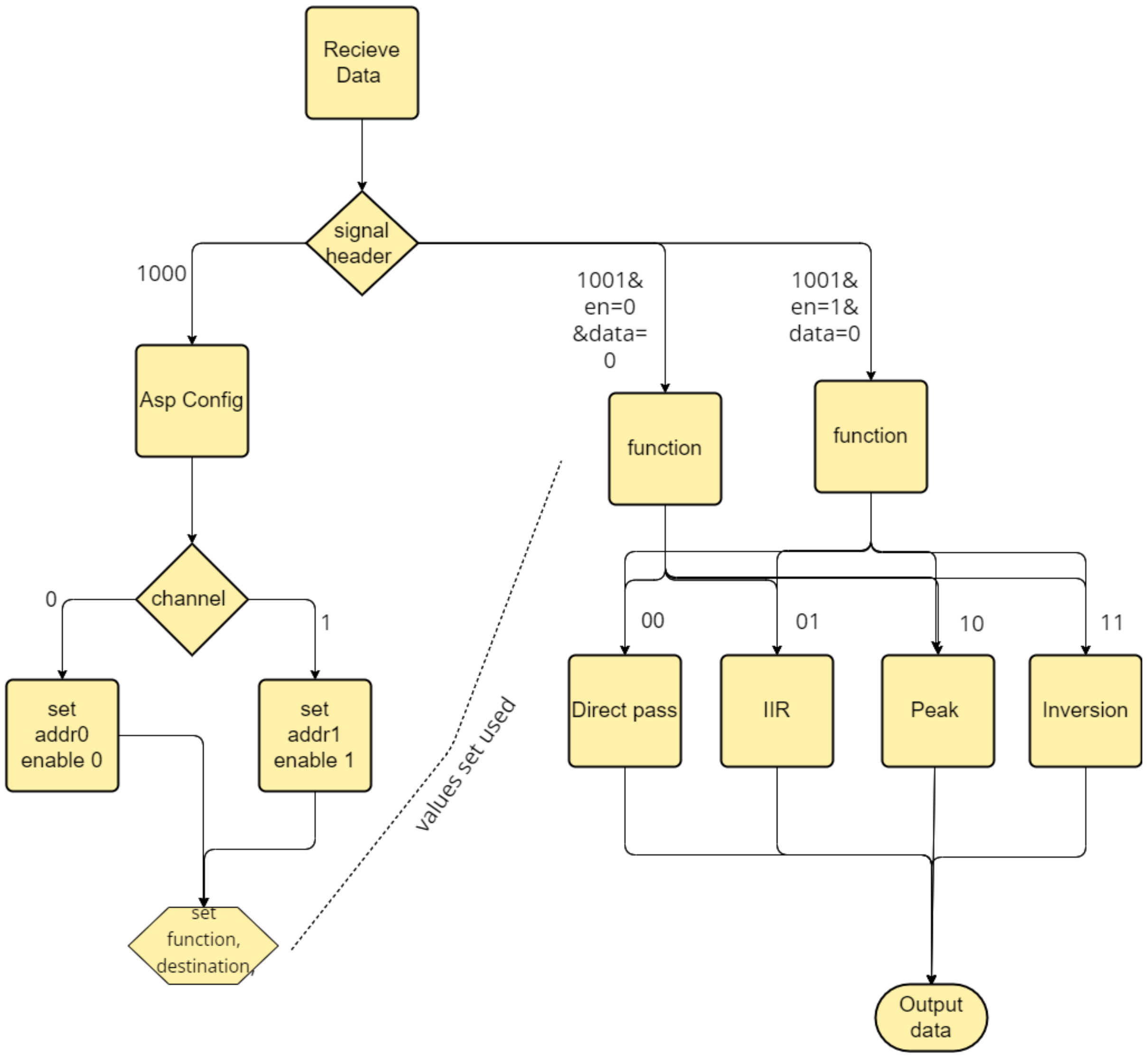
4.3. Peak Detection / Inversion ASP

This Asp can perform peak detection , inversion or iir on input stream of data depending upon mode. For peak detection it produce previous max value(peak) until the next peak is found . For inverson it subtracts new incoming data with current peak value and gives that as output . For IIR it reads first 16 bits to get two 8 bits then perform first order filtration on input .The two coefficient range from 0 to1 if input 8 bits are 256 then it is 1 if it is 0 then 0.

4.3.1. Peak ASP Operation

The ASP is configured when it receives a signal with a header of “1001”.
The output address of ASP can be configured per channel and channels can be disabled. For consistency with other ASP the Pealk Detection ASP doesn’t have two process now rather it use just single proces and combinational circuit to configure ASP .

Figure 6: Peak ASP Program Flow



4.4. FIR ASP

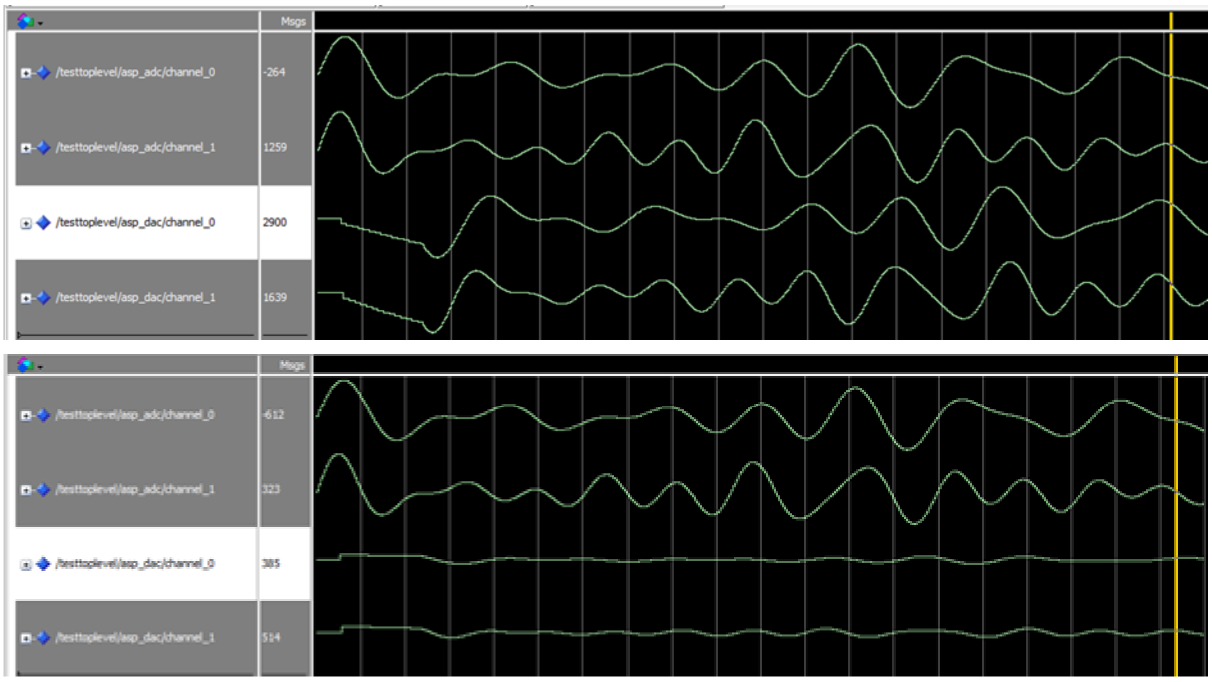
The finite impulse response (FIR) ASP performs a convolution across the impulse response and the audio signal being passed to the ASP. The impulse response is 4 values long, which makes this a 4th order FIR filter. The convolution can happen in a single clock cycle as the ASP stores the most recent audio signal values for both channel 0 and channel 1. As each of the values in the impulse response can be configured, there is a wide range of applications for this ASP.

4.4.1. FIR ASP Operation

The ASP is configured when it receives a signal with a header of "1111". The address of where the ASP outputs are sent can be configured per channel and channels can be disabled. The bottom 16-bits are used to configure the 4 values in the impulse response. Each value is a 4-bit fixed point signed value with a range from -2 to 1.75.

As seen in figure 5, the values of the impulse response can radically change the behaviour of the ASP. In 5a, the ASP is an inversion for the signal. In 5b, the ASP finds the derivative of the signal across three time steps.

Figure 7: FIR ASP Simulation. a) [-1, 0, 0, 0], b) [1, 0, 0, -1]


Msgs
/testtoplevel/asp_adc/channel_0
-264
/testtoplevel/asp_adc/channel_1
1259
/testtoplevel/asp_dac/channel_0
2900
/testtoplevel/asp_dac/channel_1
1639
Msgs
/testtoplevel/asp_adc/channel_0
-612
/testtoplevel/asp_adc/channel_1
323
/testtoplevel/asp_dac/channel_0
385
/testtoplevel/asp_dac/channel_1
514

5. ASP - ReCOP Interface

There are four registers used for communication with the multiple ASPs through the TDMA-NoC: DPCR(32), DPRR(32), DPC(1), and IRQ(1). In this iteration of the design, we increased the size of the DPRR register to 32-bits to match the size of the TDMA data port and moved the IRQ flag from DPRR to it's own register. As IRQ is now it's own register, the register CLR_IRQ was removed.

DPCR is used as output to the TDMA network and DPRR is the receiving register. These are connected to the data part of the TDMA signal and the address part is ignored by ReCOP. For ReCOP to be able to send data through the TDMA, the header bits of the data signal is parsed by a combinational circuit which then outputs the correct address for the given header. In an inverse fashion, the DPRR register is only written to when the address received from the TDMA indicates the ReCOP processor.

DPCR is written to in a single clock cycle since it receives 16-bits from RX and 16-bits from either RZ or the operand, to minimize the instructions used, the DPC flag is automatically set whenever DPCR is written to. As the ASPs do not directly connect to the ReCOP processor, the IRQ register is automatically set whenever the DPRR register is written to by the TDMA-NoC. However, these two flags are still manually cleared by the ReCOP processor later.

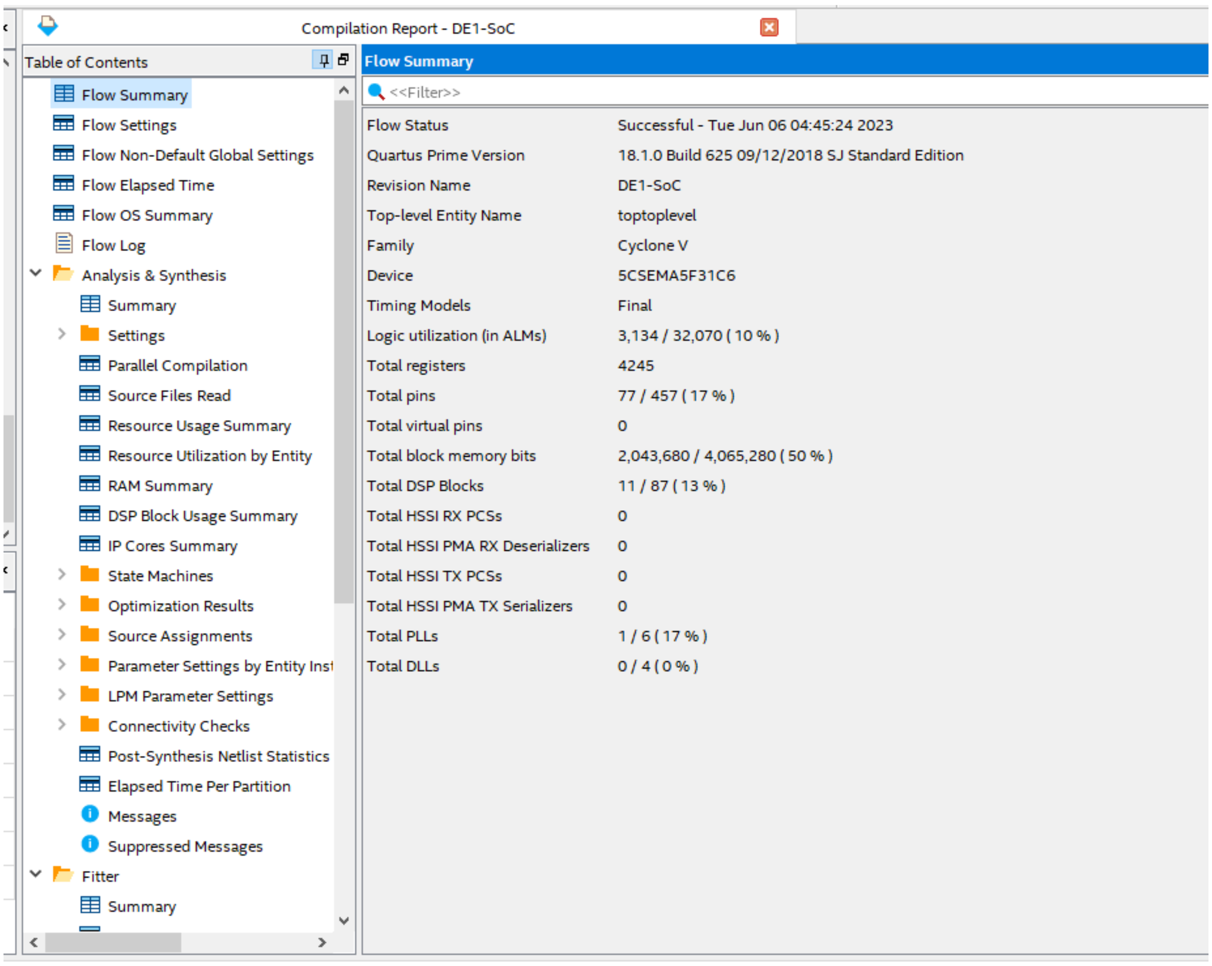
As DPC is automatically set and the address is determined by the header bits, this limits the ReCOP processor into only sending out configuration signals to the individual ASPs. While this is not a problem for the ASPs mentioned above, this may cause issues if there was an ASP that processes data and then returned the result back to ReCOP.

When both DPC and IRQ are set, instead of performing the normal fetch-decode-execution loop, the ReCOP processor goes into a different state where the received data in DPRR is passed to SOP through the register file.

6. System Verification

We verified the functionality of our system through the use of ModelSim test benches and Quartus simulation. We used Quartus Prime to synthesize our design before using the result to program an Intel FPGA board. Running our system on a prototype board verifies its applicability to real hardware

Figure 8: Quartus Synthesis Report



Compilation Report - DE1-SoC

Table of Contents

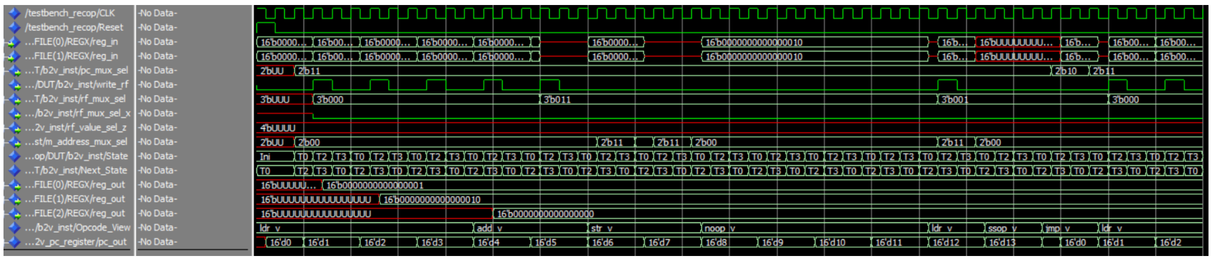
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
  - Summary
  - Settings
  - Parallel Compilation
  - Source Files Read
  - Resource Usage Summary
  - Resource Utilization by Entity
  - RAM Summary
  - DSP Block Usage Summary
  - IP Cores Summary
  - State Machines
  - Optimization Results
  - Source Assignments
  - Parameter Settings by Entity Inst
  - LPM Parameter Settings
  - Connectivity Checks
  - Post-Synthesis Netlist Statistics
  - Elapsed Time Per Partition
  - Messages
  - Suppressed Messages
- Fitter
  - Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Jun 06 04:45:24 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	DE1-SoC
Top-level Entity Name	toptoplevel
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3,134 / 32,070 (10 %)
Total registers	4245
Total pins	77 / 457 (17 %)
Total virtual pins	0
Total block memory bits	2,043,680 / 4,065,280 (50 %)
Total DSP Blocks	11 / 87 (13 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1 / 6 (17 %)
Total DLLs	0 / 4 (0 %)

Figure 9: ReCOP ModelSim Simulation


/testbench_recop/CLK
/testbench_recop/Reset
...FILE(0)/REGX/reg_in
...FILE(1)/REGX/reg_in
...T/b2v_inst/pc_mux_sel
.../DUT/b2v_inst/write_rf
...T/b2v_inst/rf_mux_sel
.../b2v_inst/rf_mux_sel_x
...2v_inst/rf_value_sel_z
...st/m_address_mux_sel
...op/DUT/b2v_inst/State
...T/b2v_inst/Next_State
...FILE(0)/REGX/reg_out
...FILE(1)/REGX/reg_out
...FILE(2)/REGX/reg_out
.../b2v_inst/Opcode_View
...2v_pc_register/pc_out
-No Data-