

[illegible]

The CPU is segmented into three modules: the **control step counter**, **combinational control gates**, and **datapath**.

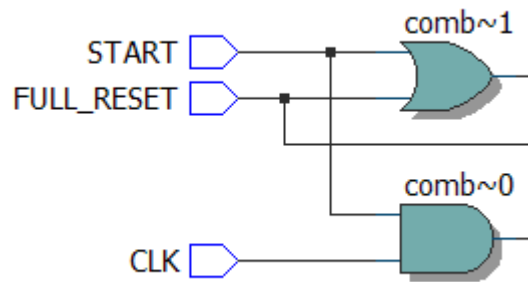
INPUT and OUTPUT PINS

INPUT PINS:

START

FULL_RESET


CLK




The **START** pin is required to be thrown in order for the **CLK** to have effect.

Both **START** and **FULL_RESET** activate the **INC** input for the control unit counter. This allows the counter to continue cycling so that on the next positive edge of the **CLK**, the CPU can reset when...

When **FULL_RESET** is active, the pulse of the **CLK** will reset the **PC** and **AC** registers and allow the CPU to read from the first line of memory once again.

—  **AC_TOCPU[7..0]**

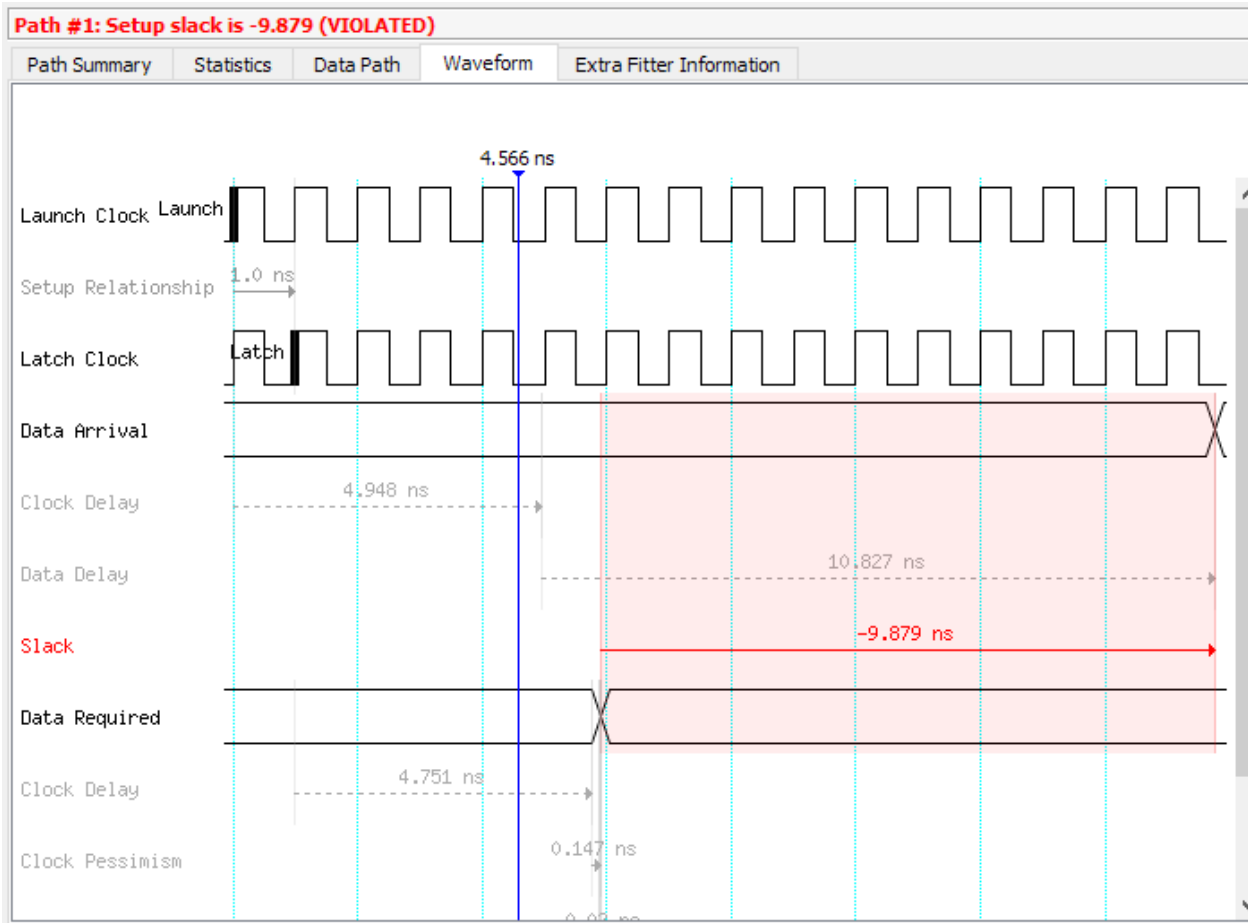
—  **PC_TOCPU[15..0]**

OUTPUT PINS:

[7:0] AC_TOCPU - The **PC** register is outputted to the red LED array on the DE2.

[15:0] PC_TOCPU - The accumulator is outputted to the green LED array on the DE2.

TIME QUEST ANALYSIS



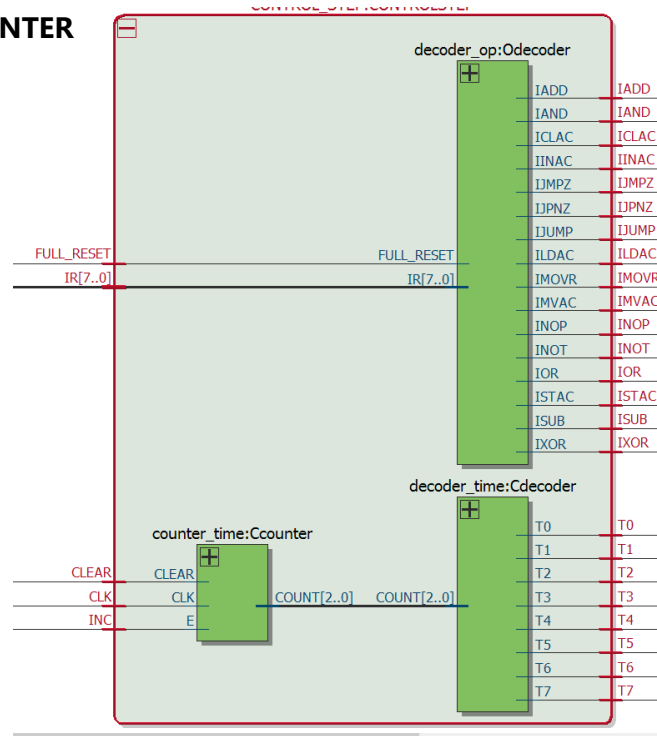
Operational frequency is determined by:

$$f = \frac{1}{Slack}$$

$$f = \frac{1}{|9.879_E - 9|}$$

$$f = 101224820 \text{ Hz or } f = \mathbf{101.2 \text{ MHz}}$$

CONTROL STEP COUNTER



Comprised of three sub-modules:

Clock counter

Responsible for enabling the counter cycle to start and reset as per instructions from the user. The counter is a simple count-up register that will count from 000 to 111 and overflow back to 000 or soft-reset to 000 depending on the required action.

Operational code decoder

This module is responsible for handling what operational code is decoded from the Instruction Register.


An example of a section of OP-code decoding is if IR has sent a 0001 signal. The operational decoder will respond with the following outputs:

INOP = 0; ILDAC = 0; ISTAC = 0; IMVAC = 0; IMOVR = 0; IJUMP = 0; IJMPZ = 0; IJPNZ = 0; IADD = 0; ISUB = 0; IINAC = 0; ICLAC = 0; IAND = 0; IOR = 0; IXOR = 0; INOT = 0;

Counter decoder

The counter decoder will take the binary number generated from the up-counter and decode its value into a corresponding T value. These T values refer to the instruction step when performing an operational code in the datapath.

COMBINATIONAL OUTPUT PINS

CONTROL_COMB:CU		
		
CLEAR		AC_BUS
IADD		AC_LOAD
IAND		ALUS2
ICLAC		ALUS3
IINAC		ALUS4
IIMPZ		ALUS5
IDPNZ		ALUS6
IDUMP		ALUS7
ILDAC		ALUS1
IMOVR		AR_INC
IMVAC		AR_LOAD
INOP		BUSMEM
INOT		DR_BUS_H
IOR		DR_BUS_L
ISTAC		DR_LOAD
ISUB		IR_LOAD
IXOR		MEMBUS
T0		PC_BUS
T1		PC_INC
T2		PC_LOAD
T3		PC_RESET
T4		R_BUS
T5		R_LOAD
T6		SOFT_RESET
T7		TR_BUS
Z		TR_LOAD
		WE

COMBINATIONAL OUTPUT PINS cont.

The operation codes are sent through the combinational output pin gates to determine what parts of the datapath receive signals. The list of output signals are displayed below:

```
PC_RESET = CLEAR;

AR_LOAD  = ( T0 | T2 | (ILDAC & T5) | (ISTAC & T5) );
AR_INC   = ( (ILDAC & T3) | (IJUMP & T3) | (ISTAC & T3) | (IJPZ & T3 & Z) |
(IJPNZ & T3 & ~Z) );

PC_BUS   = ( T0 | T2 );
PC_LOAD  = ( (IJUMP & T5) | (IJPZ & T5 & Z) | (IJPZ & T5 & ~Z) );
PC_INC   = ( T1 | (ILDAC & (T3 | T4)) | (ISTAC & (T3 | T4)) |
(IJPZ & (T3 | T4) & ~Z) | (IJPZ & (T3 | T4) & Z) );

DR_BUS_H = ( (ILDAC & T5) | (IJUMP & T5) | (ISTAC & T5) | (IJPZ & T5 & Z) |
(IJPZ & T5 & ~Z) );
DR_BUS_L = ( (ILDAC & T7) | (ISTAC & T7) );
DR_LOAD  = ( T1 | (ILDAC & (T3 | T4 | T6)) | (IJUMP & (T3 | T4)) | (ISTAC &
(T3 | T4 | T6)) | (IJPZ & (T3 | T4) & Z) | (IJPZ & (T3 | T4) & ~Z) );

TR_BUS   = ( (ILDAC & T5) | (IJUMP & T5) | (ISTAC & T5) | (IJPZ & T5 & Z) |
(IJPZ & T5 & ~Z) );
TR_LOAD  = ( (ILDAC & T4) | (IJUMP & T4) | (ISTAC & T4) | (IJPZ & T4 & Z) |
(IJPZ & T4 & ~Z) );

IR_LOAD  = T2;

R_BUS    = ( (IMOVR & T3) | (T3 & (IADD | ISUB | IAND | IOR | IXOR)) );
R_LOAD   = ( IMVAC & T3 );

AC_BUS   = ( (IMVAC & T3) | (ISTAC & T6) );
AC_LOAD  = ( (ILDAC & T7) | (T3 & (IMOVR | IADD | ISUB | IINAC | ICLAC | IAND
| IOR | IXOR | INOT)) );

MEMBUS   = ( T1 | (ILDAC & (T3 | T4 | T6)) | (IJUMP & (T3 | T4)) |
(IJPZ & (T3 | T4) & ~Z) | (ISTAC & (T3 | T4)) | (IJPZ & (T3 | T4)) );

BUSMEM    = ( ISTAC & T7 );
WE        = ( ISTAC & T7 );

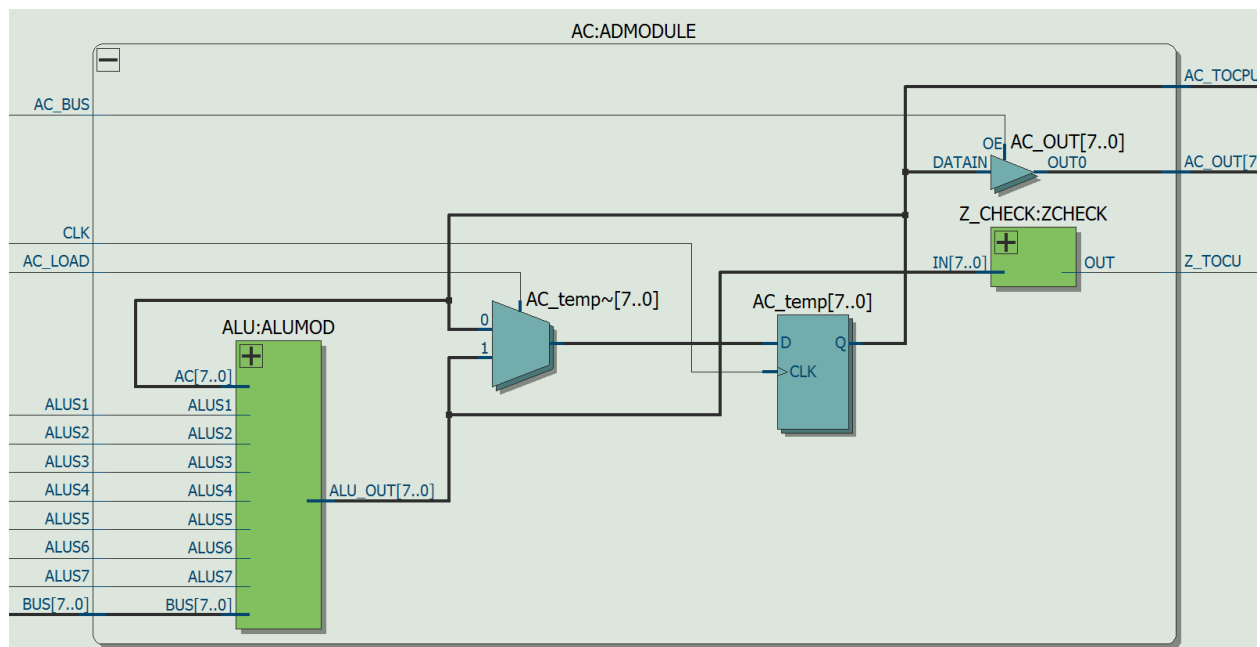
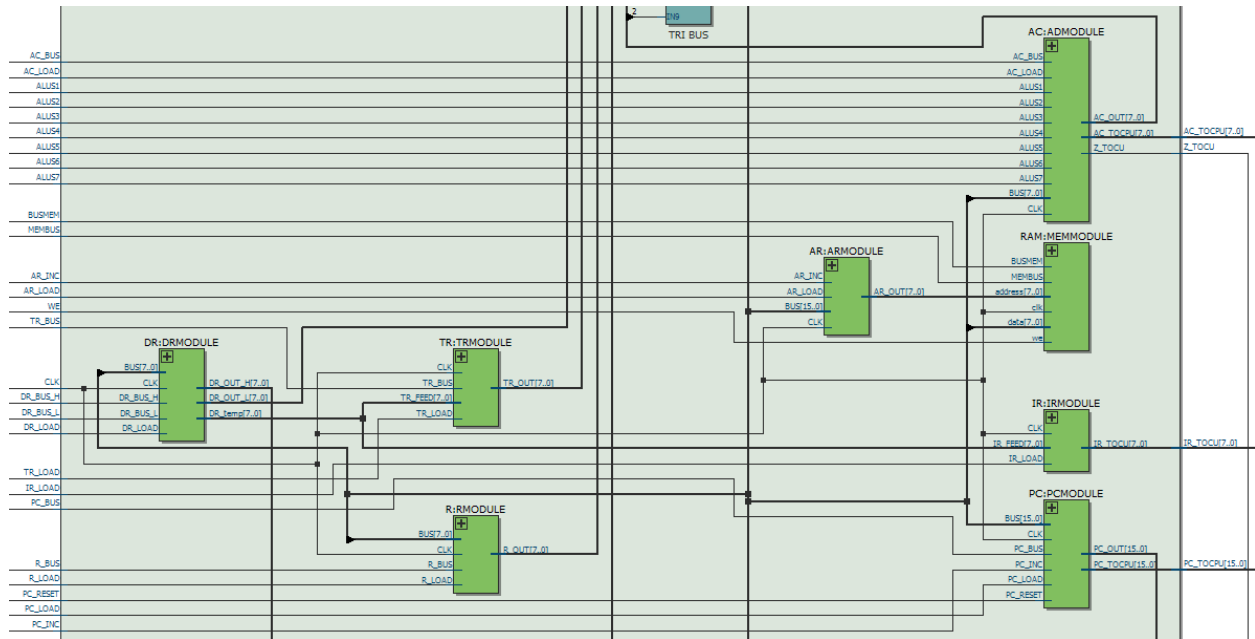
ALUS7     = T3 & (INOT | IXOR | IOR | IAND);
ALUS6     = T3 & (INOT | IOR);
ALUS5     = T3 & (INOT | IXOR);
ALUS4     = T3 & (IINAC | ISUB);
ALUS3     = (T3 & IADD) | (T7 & ILDAC);
ALUS2     = T3 & (ISUB);
ALUS1     = T3 & (IINAC | ISUB | IADD);

SOFT_RESET = ( (ICLAC & T3) | (ILDAC & T7) | (IJUMP & T5) | (T3 & (IMOVR | IMVAC))
| (ISTAC & T7) | (IJPZ & T4 & ~Z) | (IJPZ & T5 & Z) | (IJPZ & T4 & Z) |
(IJPZ & T5 & ~Z) | (INOP & T4) | (T3 & (IADD | ISUB | IINAC | IAND | IOR | IXOR | INOT)) );
```

DATAPATH

Not included in the RTL view below are the variety of tri-busses that were implemented by Quartus to handle the CPU bus. Due to the various intersection regions where the bus wires met with registers, the tri-busses were needed to handle the flow of data.

Through waveform and DE2 board testing, all registers appear to work properly and output to the bus only one register at a time. All other registers when not called to the bus output high impedance.

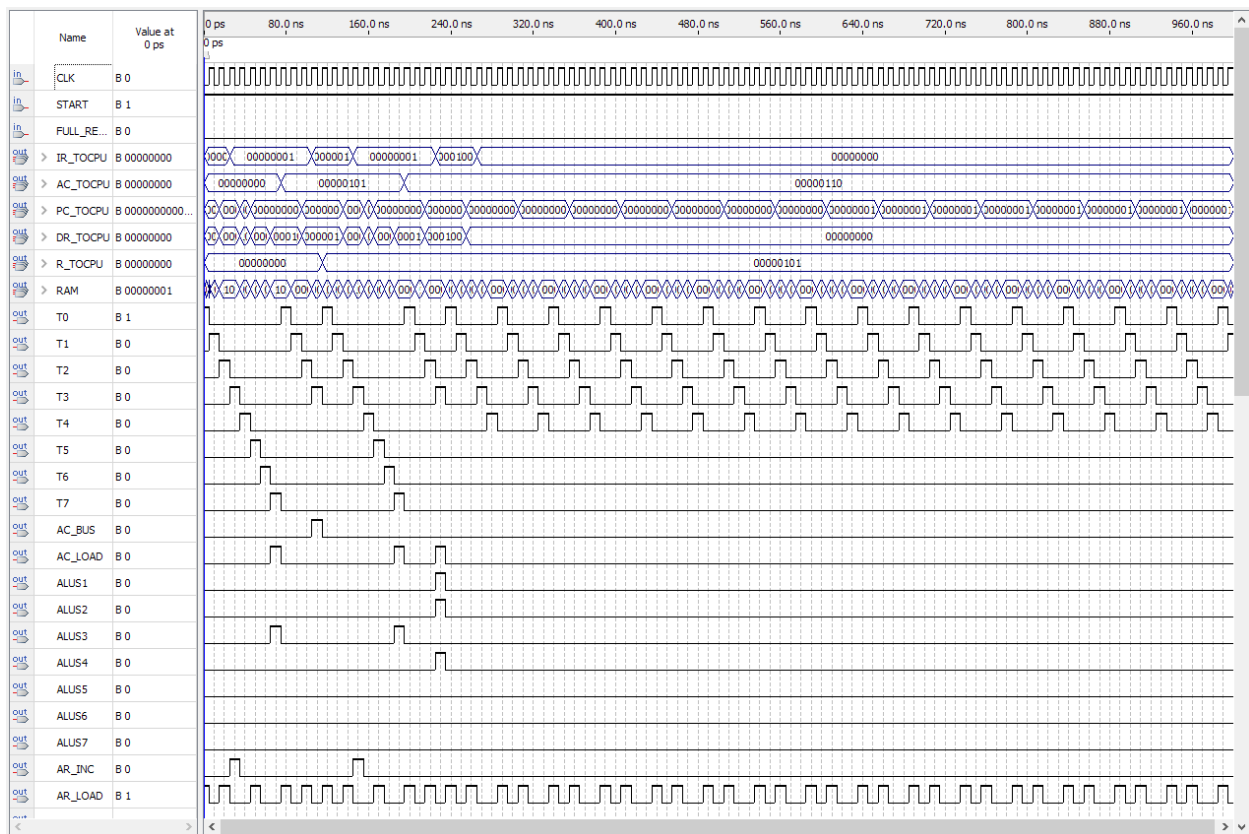


IMPLEMENTATION DIFFICULTIES

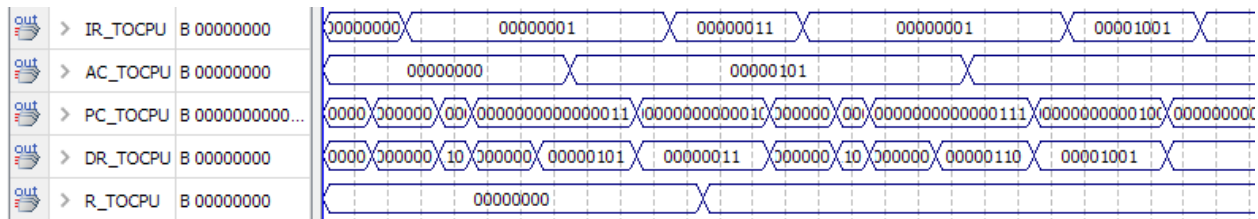
The overall construction of the CPU was ultimately inhibited by an unknown problem that seems to have resided from within the AC module inside the datapath.

Arithmetic operations such as ADD and SUB were especially problematic. When these kind of operations were required, the AC register would **not** pick up the final value looping back to the AC D flip-flops. So in other words, loading the R register and AC with appropriate values worked up until the point that the arithmetic actually had to occur.

This made me consider debugging the entire ALU sub-module to see if there was something inherently wrong. Even after reconstructing the ALU with many modifications in Verilog, I still could not figure out why the AC was not picking up the final values.



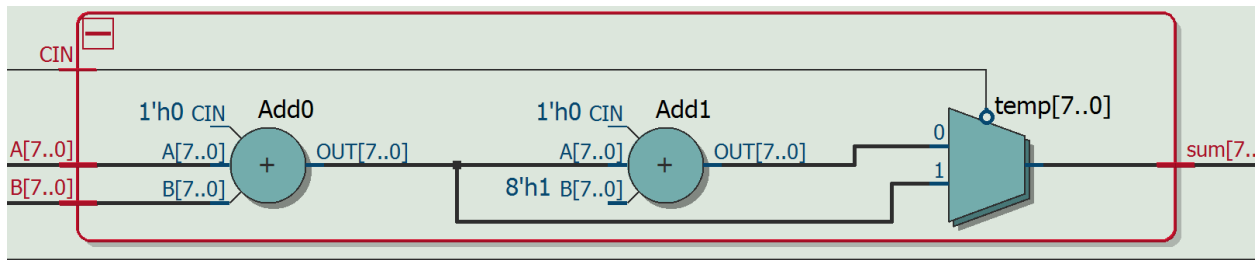
IMPLEMENTATION DIFFICULTIES cont.



To elaborate, after AC has received the second value for an operation such as $AC = AC - R$, the AC does not update even after the IR register clearly shows it is attempting subtraction.

Oddly enough, all control unit outputs are behaving as accordingly, so AC_LOAD is indeed thrown to high.

Overall, all other op-codes did appear to work properly when implemented into the meminit.mif.



This is the adder that may have been a cause of the arithmetic difficulties.

The only other resounding error that seemed to be persistent in my CPU was the PC continued to count while in START position. So regardless is INOP is being sent to the datapath, the only way for the program counter to stop was for START to be turned off.