Report: Top-Down Design Of the 8080 CPU In VHDL

Shengxin Wang, Nick Kanopoulos and Kishor S. Trivedi

July 25, 1995

Contents

List of Figures

1 Introduction

One of the main directions of modern CAD environments is the development of methodologies and tools that support a top-down, hierarchical design paradigm. This direction is necessitated, in part, by the design complexities encountered by designers today and by the need to share design data bases among large design teams. One highly touted tool in this process is the VHSIC Hardware Description Language (VHDL). The main reason for the popularity of VHDL is the fact that it has one of the best sets of constructs for hardware behavior modeling.

Early developments of VHDL tools were focused primarily on simulation and were not part of integrated tool environments. Lately, new generations of tools have emerged that offer an integrated environment where a design captured hierarchically with VHDL can be executed from functional behavior to circuit layout.

The objectives of this project were two fold. First, we intended to demonstrate a top-down hierarchical design methodology based on VHDL with a benchmark of substantial complexity. This was accomplished with the design of the Intel 8080 CPU which was performed in a top-down fashion; starting at the behavior level and ending with layout for a targeted CMOS technology. The second objective was to develop a hierarchical behavioral and structural benchmark model which can be used to perform fault-injection experiments that can be used to validate fault secure features of hardware structures. The 8080 benchmark is currently used in this capacity for the continuation of this project which is also sponsored, in part, by CACC.

The choice of the 8080 CPU for this project is based simply on the availability of structural information in the public domain for this design. The design methodology demonstrated and the fault injection methodology being developed, are independent of the particular circuit under design. However, any design attempted using the methodology presented in this project should assume the availability of some-level of structural design information at the register transfer level including timing.

The outcome of this project is a top-down hierarchical (behavioral/structural) model and an assembler for the 8080 CPU all developed in VHDL. The model was verified and validated through simulations at all levels and with the entire 8080 instruction set. All model development was performed with Mentor Graphics tools and the developed VHDL model can be used by interested users as a macrocell in the Mentor database.

2 Top-Down Design In Digital Systems

With the increasing complexity of VLSI circuits and time-to-market pressure, the bottom-up design methodologies of the 80's are no longer viable solutions, because they are tedious, slow, and error prone. A higher level top-down synthesis approach is mandatory to meet today's design needs. A design can be quickly defined using VHDL and verified through simulation. Once functionality is verified, synthesis can rapidly move the design to the gate level layout, automating a task that previously was tedious and error-prone. In addition, with a top-down design method employing VHDL synthesis, functional changes can be made rapidly and verified through simulation.

2.1 The Abstraction Hierarchy

Using VHDL, designs can be described in top-down fashion through varying levels of abstraction. The levels of abstraction in digital systems are shown in Figure 1. They are the chip level, the register level, the gate level, the circuit level, and the silicon level. Note that the abstraction hierarchy has a pyramidal shape. The broadening of the pyramid as one moves to lower levels represents the increasing amount of detail that must be managed in representing a VLSI device at that level.

Abstraction can be expressed in two domains, which are now defined as follows:

1. *Structural domain:* a domain in which a component is described in terms of an interconnection of more primitive components.

2. *Behavioral domain:* a domain in which a component is described by defining its input/output response by means of a procedure.

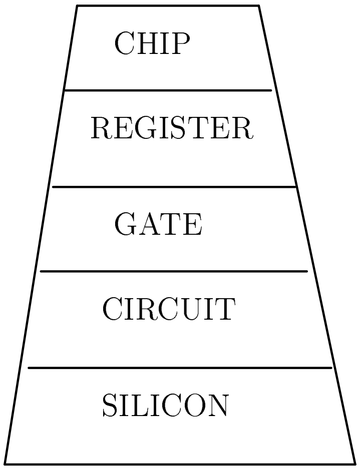

Figure 1: Abstraction levels in digital systems

2.2 Top-Down Design Process With Logic Synthesis

The flow chart in Figure 2 shows the top-down design process and the Mentor Graphics tools that can be used to implement this process. The design process includes the following steps:

1. Define the design specification and build a block diagram by partitioning the functionality.

2. Build the functional model for each top-level component. The top-level is the top level of what will be a multi-level design hierarchy. And one can describe each component in the top-level using a VHDL model, schematics, or a combination of both. The VHDL models need to be "synthesizable" RTL-level models. This model creation in the Mentor Graphics CAD framework can be performed using Design Architect or just a plain editor.

3. Verify the functionality of the top-level model by simulation. If any problems are revealed by simulation, the VHDL model is edited and

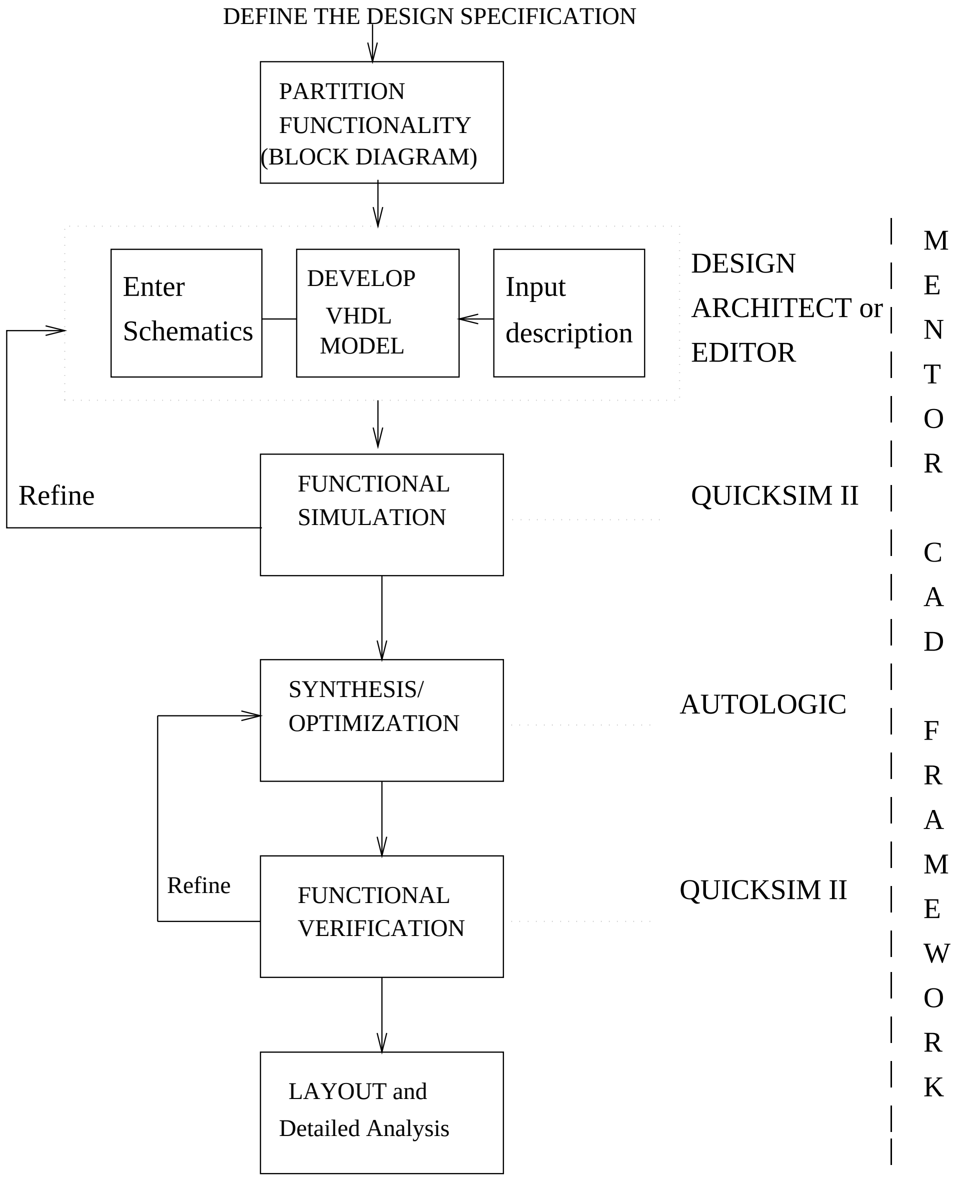

Figure 2: Top-Down Design Process With Logic Synthesis

corrected. QuickSim (Logic Simulator of Mentor Graphics) offers the capabilities needed to perform the verification.

4. Once the VHDL design is modeled correctly and verified, the Autologic (Synthesis tool for Mentor Graphics) environment can be used to synthesize and optimize the design.

5. Verify the behavior of the synthesized gate-level design to determine whether the synthesized design operates correctly. Again Quicksim is used to simulate the design.

The process of developing a synthesizable VHDL model in step 2 above includes the following steps:

- Develop an abstract behavioral VHDL model.
- Verify the abstract behavior of the model.
- Decompose the behavioral model into a "synthesizable" RTL-level model.
- Verify the behavior of the refined model.

3 Important Features of VHDL

Top-Down design requires a common medium, a requirement satisfied by the VHDL language. The basic descriptions of the syntax and constructs of VHDL could be found in many publications [1, 2, 8, 5]. Some important features of VHDL make it very effective for modeling a logic design. These features are briefly described to provide context for the project described in this report.

3.1 Classes of Objects

There are three classes of objects in VHDL: constants, variables and signals. An object is created when it is declared. The difference between signals and variables is that signals have a time dimension and direct hardware correspondence.

3.2 Attributes

Signal attributes are particularly important in modeling. Some examples are:

1. S'LAST_VALUE is the previous value of S immediatedly before the last change of S. It is very useful for checking changes in a signal.
2. S'STABLE(T) is of type BOOLEAN. It is true if S has been stable for last T time units.
3. S'DELAYED(T) is the value of S, T time units earlier.

Another useful set of attributes are those associated with arrays. For example, suppose that an array variable was defined as follows:

variable A: BIT_VECTOR(0 to 15);

Then the following attributes can be indicated:

Attribute	Value
A'RANGE	0 to 15
A'LEFT	0
A'RIGHT	15

3.3 Modeling concurrency

The modeling of logic circuits has a requirement that is not shared by many other modeling applications; that is, the model must include provision for concurrency of execution. This is because logic signals flow in parallel. Fig. 3 illustrates this concept. Three logic blocks are shown. If one assumes that input set 1 and set 2 are activated simultaneously, logic blocks 1 and 2 will be activated together. Logic block 3 will be activated as soon as either of the outputs from logic block 1 (Z1) or from logic block 2 (Z2) change. While signals are propagating through logic block 3, new input signal changes can be propagating through block 1 and block 2. Thus signal flow can take place through all blocks simultaneously.

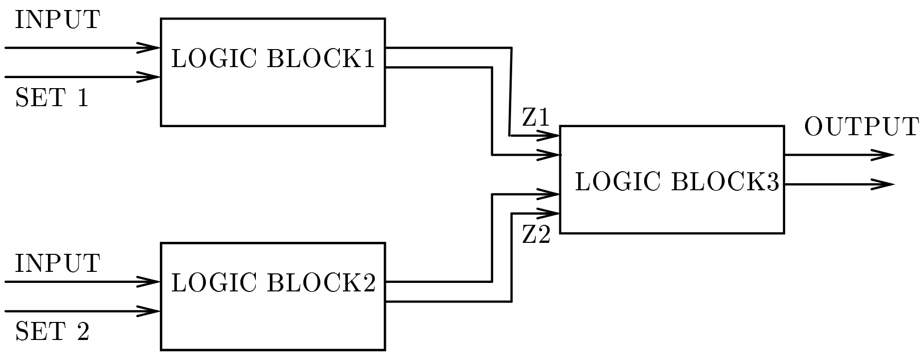

Figure 3: Logic block structure

```
LOGIC_BLOCK1: Process (Input Set 1)
              begin
              ———-
              end process LOGIC_BLOCK1 ;
LOGIC_BLOCK2: Process (Input Set 2)
              begin
              ———-
              end process LOGIC_BLOCK2;
LOGIC_BLOCK3: Process (Input Set 3)
              begin
              ———-
              end process LOGIC_BLOCK3;
```

Figure 4: Three concurrent processes

The hardware description language must have a mechanism for modeling this operation. In VHDL, this requirement is handled by the process construct. Each process represents a block of logic, and all processes execute in parallel. In VHDL, a process is activated when a signal in its sensitivity list changes. Figure 4 shows the corresponding VHDL process structures for the logic block structure in Figure 3. Note that the sensitivity list for the process contains, in general, the input signal set for the logic block.

3.4 Signal Assignment

Signals are used to transmit information between processes. A signal assignment statement is the major important behavior statement in the language. There are two types of delay that can be applied when assigning a time/value pair into the driver of a signal: inertial and transport. They have the following forms:

Signal name <= Value **after** time-expression;

Signal name <= **transport** value **after** time-expression;

The first assignment statement implies inertial delay. Inertial delay is used for devices that do not respond unless a value on its input persists for a given amount of time, and it is useful in modeling devices that ignore spikes on their inputs. The default delay type in VHDL is inertial. In the second case, the transport delay is specified. All changes on input will propagate to output regardless of how long the changes stay at the new level. Transport delay would more likely be employed at higher levels of abstraction.

3.5 Process Model Graph

A graph representation for high-level models is very useful, because it allows us to model complex signal flow within such models. Process model graphs represent a partitioning of the model, with each node representing a function. This partitioning can also be applied into the nodes, so a node could be decomposed into subfunctions such as node D in the example flow graph illustrated in Figure 5. When modeling in VHDL, each such subfunction can be implemented by a VHDL process. Arcs are used to denote signal passage between the process models. Each arc is labeled with a designator of the signal which transmits from one process to another.

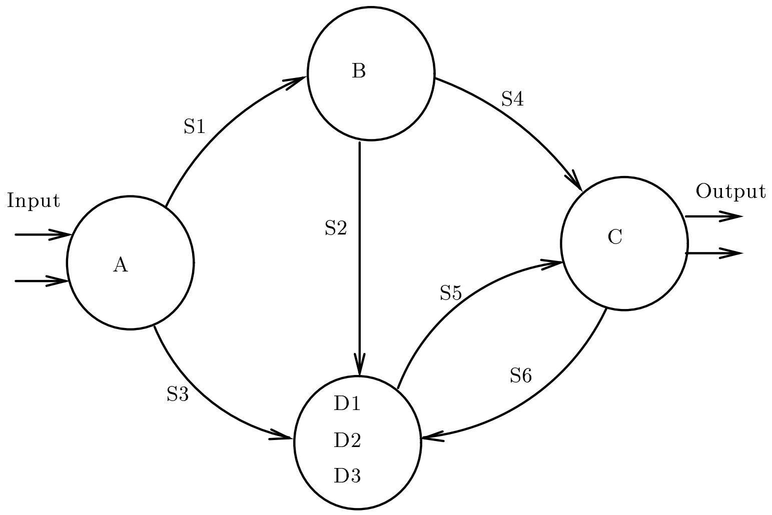

Figure 5: Process Model Graph

The process model graph clearly illustrates the structure of a high level model in the behavioral domain. All models are made more understandable by the use of process model graphs. And this approach was effectively used to illustrate clearly the signal flow very clearly in the VHDL design of the 8080 CPU.

3.6 Guidelines for synthesizable VHDL

There are certain guidelines that need to be followed when using VHDL for synthesis. They encompass syntax requirements, constructs constrains, modeling methods, and design methods. These guidelines exist because synthesizable VHDL is actually a large subset of the entire VHDL language. Some of these guidelines are summerized as follows:

1. There could be only **one clock** for each process. Attributes like EVENT, LAST_VALUE are only for clock signals. AUTOLOGIC VHDL supports clocks defined of type bit, boolean, qsim_state(qim_logic package), or std_ulogc(std_logic_1164 package).

2. For an unclocked process, when assigning to variables with conditional assignments, one needs to assign the variables all the time since they are not going to be synthesized into latches. Otherwise unsynthesizable internal state errors will be issued. But if the process is a clocked one, the variables are going to be synthesized into flip-flops.

3. Signals are not allowed to appear on both sides of the assignments in one process unless they are in the sensitivity list of that process. Otherwise, unsynthesizable internal state errors will be issued. Care must be taken when adding signals to a process sensitivity list, since the process gets activated each time if there is any change in its sensitivity list. The behavior could then be altered dramatically.

4. INOUT port could only be of the BUS type. Tri-State Logic needs to be handled carefully. There are two ways for handling tri-state logic. One is to use the process statement, signals of the BUS kind, and the null waveform assignment. When a signal is assigned a null waveform, it is disconnected.

5. For more information about synthesizable VHDL, refer to "Autologic VHDL Synthesis Guide" by Mentor Graphics [3]. These constrains have a very significant effect on the model's design. A "synthesizable" model sometimes needs to be reconstructed to guarantee that it has the right behavior at the same time.

4 The 8080 Central Processing Unit

The central processing unit (CPU) controls the functions performed by almost all other components of a computer system. The CPU must be able to fetch instructions from memory, decode their binary contents, and then execute them. In addition, the CPU should be able to recognize and respond to certain external control signals, such as INTERRUPT and WAIT requests.

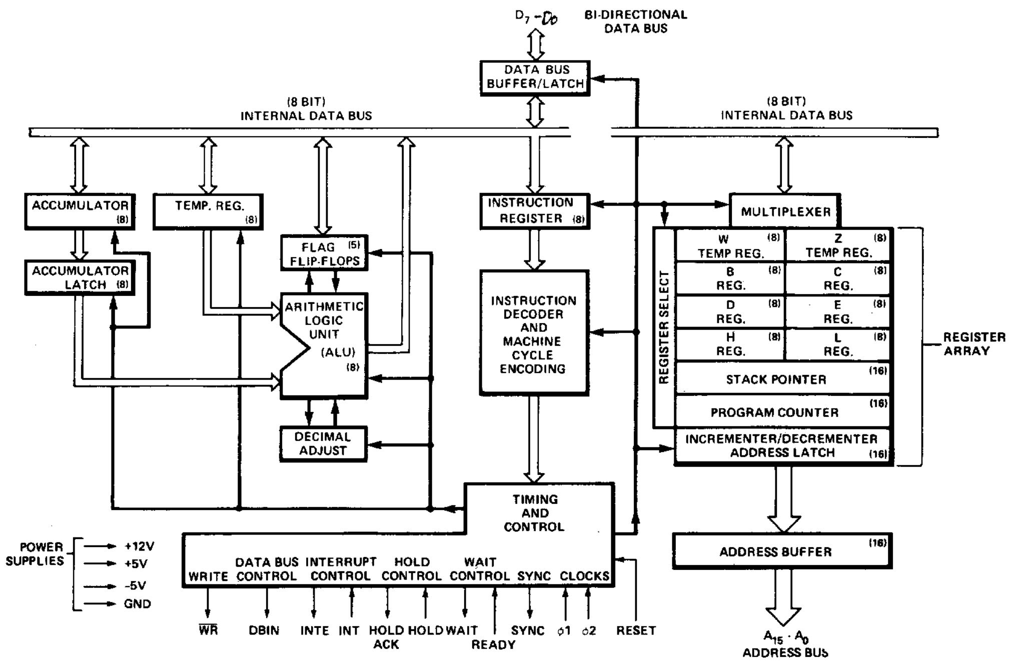

Figure 6: 8080 CPU Functional Block Diagram

4.1 Architecture of the 8080 Central Processing Unit

Figure 6 illustrates the functional blocks within the 8080 CPU. The 8080 CPU consists of the following functional units:

- Register Array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bi-directional, 3-state data bus buffer

The register section consists of a static RAM array organized into six 16-bit registers, including the program counter (PC), stack pointer (SP) and six 8-bit general purpose registers arranged in pairs referred to as B,C; D,E; H,L and a temporary register pair called W, Z.

The ALU contains an 8-bit accumulator, an 8-bit temporary accumulator, an 8-bit temporary register, and a 5-bit flag register with flags: zero, carry, sign, parity and auxiliary carry. Arithmetic, logical and rotate operations are performed in the ALU. ALU is fed by the temporary register (TMP) and the temporary accumulator (ACT) and carry flip-flop. The result of the operation can be transferred to the internal bus or to the accumulator; the ALU also feeds the flag register.

The first byte of an instruction (containing the opcode) is loaded from the internal bus to the 8-bit instruction register during an instruction fetch. The contents of the instruction register are in turn, available to the instruction decoder. The output of the decoder, combined with various timing signals from the control unit, provides the control signals for the register array, ALU, and data buffer blocks. In addition, the outputs from the instruction decoder and external control signals feed the timing and state control unit which generates the timing and state status signals.

The control circuitry is the primary functional unit within a CPU. Using two-phase clock inputs, the control circuitry maintains the proper sequence of events required for any processing task. After an instruction is fetched and decoded, the control circuitry issues the appropriate signals (to units both internal and external to the CPU) for initiating the proper processing action. The control circuit is capable of responding to external signals, such as an INTERRUPT, HOLD, WAIT, and RESET requests. An interrupt request will cause the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program. A wait request is often issued by a memory or I/O element that operates slower than the CPU. The control circuitry will idle the CPU until the memory or I/O port is ready with the data. Some peripheral devices are capable of transferring information to and from memory much faster than the processor itself can accomplish the transfer. So the hold provision enables Direct Memory Access operations (DMA), which improves the throughput of a processor. The processor must temporarily suspend its operation during such a transfer, to prevent conflicts that would arise if processor and peripheral device attempted to access memory simultaneously. A reset signal is used to start-up the CPU.

4.2 The 8080 Processor Cycle

An instruction cycle is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction is extracted from memory and deposited in the CPU's instruction register. During the execution phase, the instruction is decoded and translated into specific processing activities.

Every instruction cycle consists of one to five machine cycles. A machine cycle is required each time the CPU accesses the memory or an I/O port. The following ten different types of machine cycles may occur within an instruction cycle while no one instruction cycle will consist of more than five machine cycles:

1. FETCH(M1)
2. MEMORY READ
3. MEMORY WRITE
4. STACK READ
5. STACK WRITE

6. INPUT
7. OUTPUT
8. INTERRUPT
9. HALT
10. HALT.INTERRUPT

The machine cycles that actually occur in a particular instruction cycle depend upon the kind of instruction executed. However, the first machine cycle in every instruction cycle is always a FETCH cycle. Each machine cycle consists of three to five states. A state is the smallest unit of processing activity and is marked by a clock cycle.

To summarize, each clock period marks a state, three to five states constitute a machine cycle, and one to five machine cycles comprise an instruction cycle.

5 VHDL Design of the 8080 CPU

5.1 Process Model Graph of the 8080 CPU

The system VHDL model is interconnected by the following components:

1. ALU
2. Control_State
3. Control_Signal
4. InstructionDecoder
5. Register Array

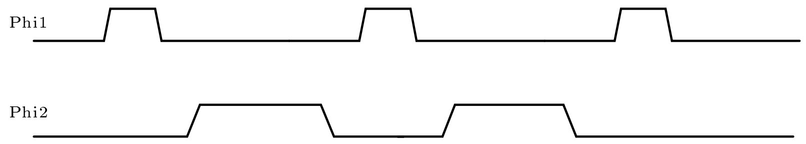

Figure 7: Two-Phase Nonoverlapping Clock

In addition to those main processor model components, it is necessary to have a package that contains data types and functions that perform signal increment and decrement, and signed 2's complement conversion.

The 8080 CPU needs to be driven by a two-phase clock oscillator. And the two phases of the clock generated by the oscillator are nonoverlapping and are labeled as phi1 and phi2. Figure 7 illustrates the two phase nonoverlapping clock. All processing activities for the VHDL model of the 8080 are actually referred to the events of these two clock phases.

Figure 8 shows the Process Model Graph of the VHDL design of the 8080 CPU. In what follows, the function of the VHDL model of the 8080 is explained by referring to the Process Model Graph in Figure 8. It is the rising edge of clock pulse phi1 that divides each machine cycle into states (represented by signal T) in the Control_State process. With this state signal, at the falling edge of clock phi1, the InstructionDecoder process activates its output signals such as enabling and multiplexing signals flowing to ALU and Register Unit processes, as well as the machine cycle status signals (M and ST) and other instruction flag signals (FLAGS) flowing to Control_Signal process. The Control_Signal process then triggers the control signals such as SYNC, DBIN, and others at the rising edge of clock pulse phi2 with all the machine cycle status signals and instruction flag signals that have already been set by the InstructionDecoder process at the falling edge of phi1. The INTERNAL_BUS (8 bit) is used to transmit data from ALU to Register Unit and vice-versa; While the DATABUS (8 bit) is used to transmit data from outside to CPU and vice-versa. The INTERNAL_BUS reads in transmitted data from registers in either ALU or Register Unit whenever its enabling signals are set by the InstructionDecoder process at the falling edge of phi1. The DATABUS gets data from outside whenever the DBIN signal is set,

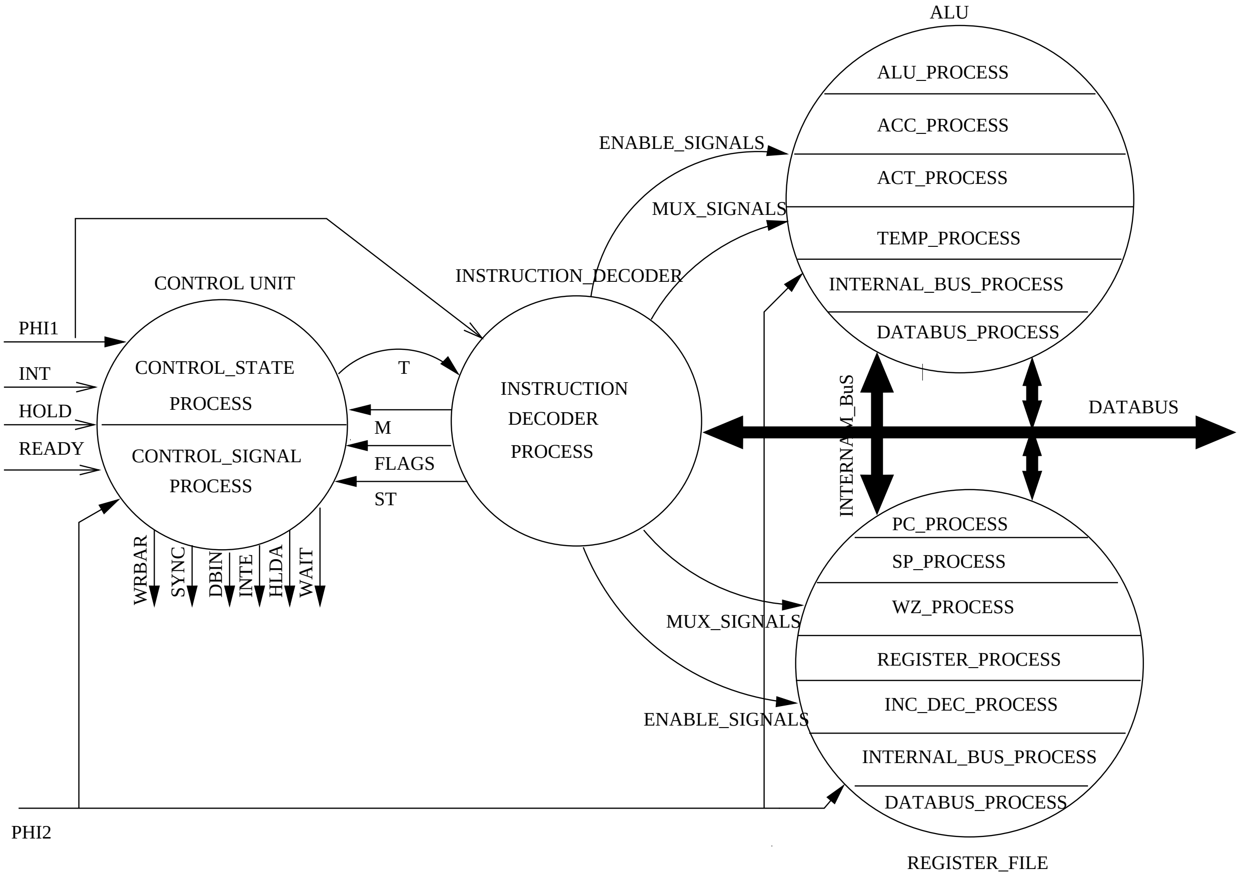

Figure 8: 8080 CPU Process Model Graph

which is done by the Control_Signal process at the rising edge of phi2. All the registers in both ALU and Register Unit actually read the transmitted data at the falling edge of clock phi2 from INTERNAL_BUS and DATABUS, where the data has been available since the falling edge of clock phi1 or the rising edge of clock phi2. The fact that these processing activities get initiated at different states of the two-phase clock guarantees the right order of complex signal and data flow within the VHDL model of the 8080 CPU.

5.2 Model of the Control Unit

The control unit is implemented as two processes namely Control_State and Control_Signal. Control_State is activated by the rising edge of phi1, which controls the state transition and sets the control signals that are related to the leading edge of phi1. Control_Signal issues the external control signals which need to be activated at the rising edge of phi2.

The state transition diagram in Figure 9 shows how the 8080 proceeds from state to state in the course of a machine cycle. The diagram also shows how the external signal lines such as READY, HOLD, and INTERRUPT are sampled during the machine cycle, and how the conditions on these lines may modify the basic transition sequence.

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T1, T2, T3, T4, T5 or TW). The SYNC signal identifies the first state (T1) in every machine cycle, and is related to the leading edge of phi2. The following transition occurs at the next phi2 pulse.

During T1, the processor sends an address to memory. There is an opportunity for the memory to request a WAIT. This is done by pulling the processor's READY line low. The READY signal is sampled during T2 by the processor. The processor responds to a wait request by entering an idle state (TW) at the end of T2, rather than proceeding directly to state T3. This gives the memory time to respond to the addressed data request. Entry into TW state is indicated by a WAIT signal from the processor, acknowledging the memory's request. And the low-to-high transition on the WAIT line is triggered by the rising edge of phi1 and occurs at the actual entry of the TW state.

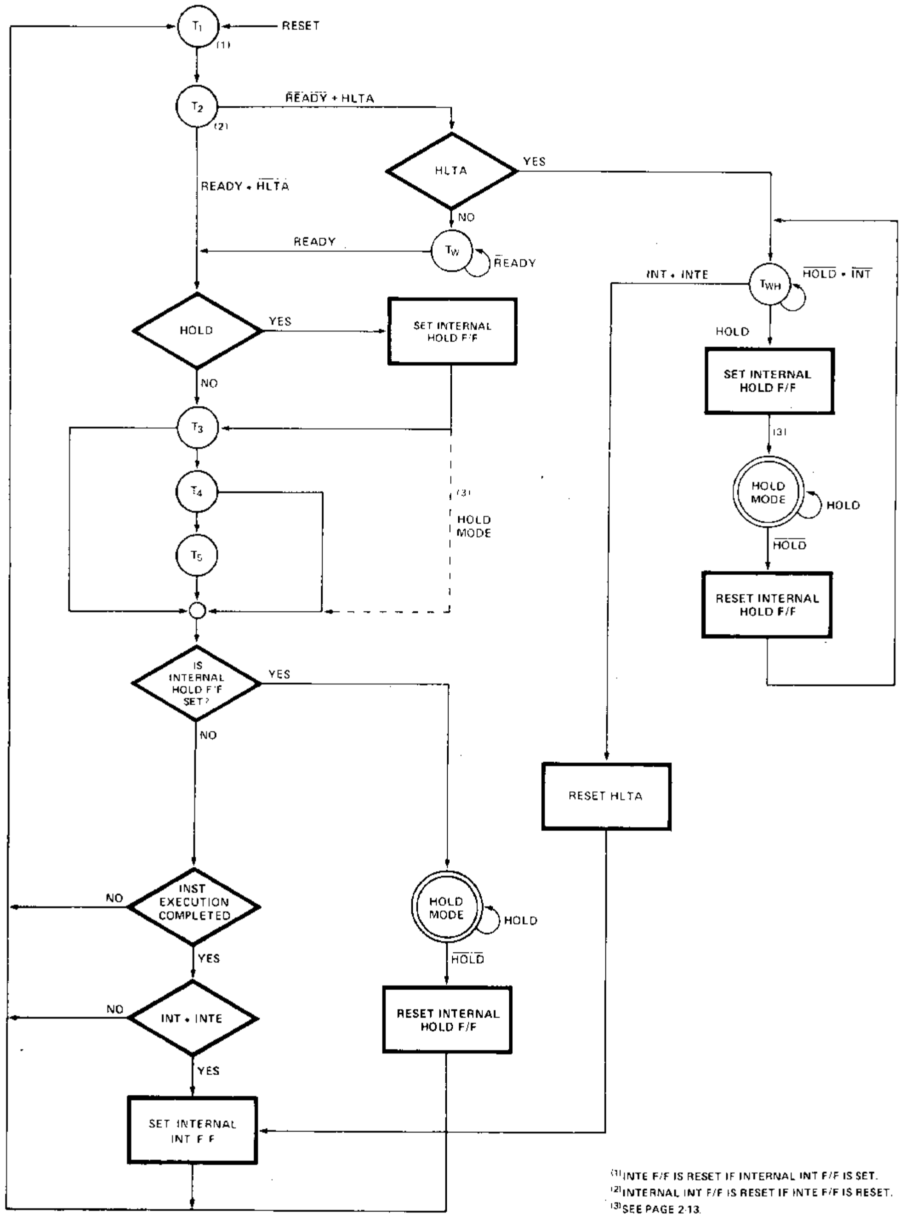

Figure 9: CPU State Transition Diagram

When a halt instruction (HLT) is executed, the CPU enters the halt state (TWH) after state T2 of the next machine cycle. There are three ways for the 8080 to exit the halt state:

- A high on the RESET line will always reset the 8080 to state T1; RESET also clears the program counter.
- A HOLD input will cause the 8080 to enter the hold state. When the hold line goes low, the 8080 re-enters the halt on the rising edge of the next phi1 clock.
- An interrupt will cause the 8080 to exit the HALT state and enter T1 on the rising edge of the next phi1 clock. NOTE: The interrupt enable (INTE) flag **must** be set when the halt state is entered.

The HOLD request line is sampled during T2. An internal hold latch is set as a coincidence of the READY, the HOLD, and the phi2 signals. Setting the latch enables the subsequent rising edge of the phi1 clock to trigger the HLDA (hold acknowledgement signal) output and the CPU enters the HOLD mode. During an acknowledged HOLD, the address and data busses are under control of the peripheral which originated the request, enabling it to conduct memory transfers without processor intervention. However, internally, certain functions may continue. If a HOLD REQUEST is acknowledged at T3, and if the processor is in the middle of a machine cycle which requires four or more states to complete, the CPU proceeds through T4 and T5 before coming to a rest. Not until the end of the machine cycle is reached will processing activities cease. Internal processing is thus permitted to overlap the external DMA transfer, improving the efficiency of the system. After the HOLD REQUEST is terminated, the HLDA output returns to a low level following the leading edge of the next phi1 clock pulse.

A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high. The interrupt (INT) request arriving during the time that the interrupt enable line (INTE) is high, acts in coincidence with the phi2 clock to set the internal interrupt latch. This event takes place during the last state of the instruction cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be processed.

All processor machine cycles consist of at least three states: T1, T2 and T3. If the processor has to wait for a response from the peripheral or the memory with which it is communicating, then the machine cycle may also contain one or more TW state. After the T3 state, the processor will terminate any machine cycle as soon as its processing activities are completed, rather than proceeding through the T4 and T5 states every time. Thus the 8080 may exit a machine cycle following the T3, the T4, or the T5 state and proceed directly to the T1 state of the next machine cycle.

In reference to other external control signals, the DBIN signal is used externally to enable data transfer. Machine cycles in which DBIN is available include: FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of phi2 during T2 and terminated by the corresponding edge of phi2 during T3. Any TW phases intervening between T2 and T3 will therefore extend DBIN by one or more clock periods. A WRBAR output control signal is generated by the 8080 CPU for the synchronization of external transfers, during those machine cycles in which the processor outputs data. Those include MEMORY WRITE, STACK WRITE, and OUTPUT. The negative-going leading edge of WRBAR is referenced to the rising edge of the first phi1 clock following T2. It remains low until retriggered by the leading edge of phi1 during the state following T3. Note that any TW states intervening between T2 and T3 at the output machine cycle will necessarily extend WRBAR, in much the same way that DBIN is affected during data input operations.

5.3 Model of the InstructionDecoder

The InstructionDecoder process is implemented by using three nested **case-when** statements for different machine cycles, different states and different instructions. All the events that actually take place depend on the kind of instruction involved, and on the particular state and machine cycle within the instruction cycle. Those events include issuing the appropriate enabling and multiplexing signals for initiating processing actions in the ALU and Register Array, doing machine cycle encoding, and setting the instruction flags that will flow into Control_Signal process. The InstructionDecoder is a clocked process in which all the events are activated at the falling edge of clock phi1 with the state status signal initiated at the rising edge of phi1 by the Control_State process. So there is also enough time for the Control_Signal process to get the instruction flag and machine cycle status signals from the InstructionDecoder process to activate the external control signals.

5.4 Model of the Register Array

There are several processes for the model of the register array unit:

1. A process to increment or decrement the Program Counter (PC), Stack Pointer (SP), or other register pairs. It is not a clocked process and gets activated by any change in the process's sensitivity list.

2. The register pair processes like Register_Proc, PC_MUX_Proc, SP_MUX_Proc, and WZ_MUX_Proc, which behave pretty much like the register hardware primitives. The transferred data are read into the registers at the falling edge of phi2 when their enabling signals are set to high by the InstructionDecoder process. And their inputs are multiplexed according to the multiplexing signals triggered also by the InstructionDecoder process.

3. The address latch process, ADDRESS_LATCH_Proc, is enabled to read in the address if its enable signal is set to high, and the process gets activated by any change in its sensitivity list.

4. The INTERNAL_BUS and DATABUS are both Tri-State Logic (multiply driven signals), they need to be resolved very carefully. The INTERNAL_BUS_OUT_Proc and DATABUS_OUT_Proc processes assign null waveform to the BUS kind signals whenever they are not driving the bus signals. When a bus signal is assigned a null waveform, it is disconnected from the driver.

5.5 Model of the ALU Unit

The ALU model has the following components:

1. ALU_Proc process which does arithmetic, logic and rotation operations, is not a clocked process and gets activated by any change in the process's sensitivity list.

2. The register processes including TEMP_MUX_Proc and ACC_MUX_Proc, are each enabled by the enabling signals from the InstructionDecoder process. The data transfers are triggered by the falling edge of the clock phi2 and are multiplexed by the multiplexing signals.

3. The process of ACT latch, ACT_MUX_Proc is actually enabled when the enabling signal is set to high by the InstructionDecoder process, and is activated by any change in its sensitivity list.

4. INTERNAL_BUS_OUT_Proc and DATABUS_OUT_Proc processes are similar to those described in the Register Array Model.

Note that the usage of several processes in the Register Array Model and also the ALU Model has to do with the concurrent events among these processes, since the concurrent activities can not be implemented within one process. In addition, it relates to the synthesis purpose, since there are some constrains for a synthesizable VHDL design, which have been discussed briefly in section 3.6.

6 Simulation and Synthesis of the VHDL model of the 8080 CPU

6.1 Model test vectors and input files

In order to verify the functionalities of the 8080 CPU model, two test files were prepared. The first file consists of binary numbers which represent the stimuli including a clock signal, and external request signals such as RESET, READY, HOLD, INT. The stimuli are added to the 8080 CPU system by using a VHDL file which reads in the stimuli from the binary stimuli file and then connects these stimuli to the 8080 system model. Another test file is the entire instruction set of the 8080 CPU in assembly language. The instructions are added to the 8080 CPU system by using a VHDL file which reads in instructions in assembly language, converts them to binary codes, and then adds these binary codes to the 8080 system model. This VHDL file acts like an assembler in the sense that it converts assembly language into binary codes for the entire instruction set of 8080 CPU. This assembler was also developed in VHDL during the course of this project. It greatly facilitates the process of debugging if the instruction test file for the entire instruction set is in assembly language instead of binary codes.

6.2 Simulation Results

1. The state transition is verified for each of the instructions of the 8080 CPU to make sure that it has the the right number of T states for every machine cycle, and the right number of machine cycles for each instruction cycle. An output sample is shown below:

```
M1_T1          -- State T1 of the first machine cycle(M1)
M1_T2
M1_T3
M1_T4
11111011       -- Instruction binary code
M1_T1
M1_T2
M1_T3
M1_T4
01110110
M2_T1
M2_T2
M1_T1
M1_T2
M1_T3
M1_T4
11000111
M1_T5
M2_T1
M2_T2
```

```
M2_T3
M3_T1
M3_T2
M3_T3
M1_T1
M1_T2
M1_T3
M1_T4
00000001
M2_T1
M2_T2
M2_T3
M3_T1
M3_T2
M3_T3
```

2. The waveform of external control signals like SYNC, DBIN, and WR-BAR are traced for the first several instructions being executed to make sure that they get set and reset properly. Figure 10 shows the waveform of these signals.

3. The values for all the registers in both ALU and Register Array Unit are checked at the end of each of the instructions to make sure that each instruction gets executed in the way it is supposed to. A sample output is shown below:

```
IR 11111011          -- Instruction Register
PC 0000000000000001 -- Program Counter
SP 0000000000000010 -- Stack Pointer
Reg B 00000000      -- Register B
Reg C 00000000      -- Register C
Reg D 00000000      -- Register D
Reg E 00000000      -- Register E
Reg H 00000000      -- Register H
Reg L 00000000      -- Register L
```

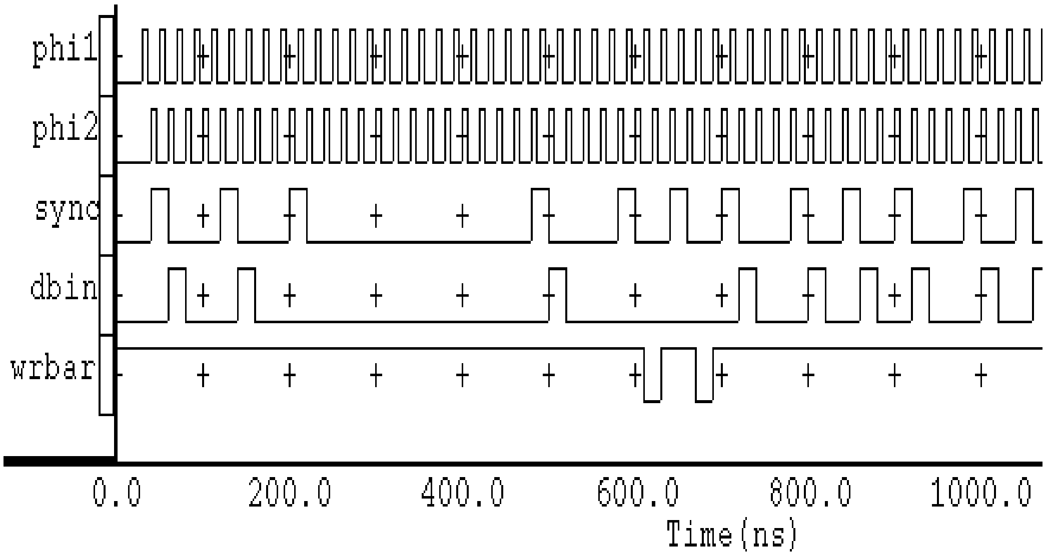

Figure 10: The Waveforms of the External Control Signals

```
ALU   0             -- ALU ouptut
ACC   00000000      -- Accumulator Register
TEMP  00000000     -- Temporary Register
FLAGS: ZO, P, S, CY, AC 0 0 0 0 0  -- Flag Register in ALU


IR 01110110
PC 0000000000000010
SP 0000000000000010
Reg B 00000000
Reg C 00000000
Reg D 00000000
Reg E 00000000
Reg H 00000000
Reg L 00000000

ALU   0
```

```
ACC   00000000
TEMP  00000000
FLAGS: ZO, P, S, CY, AC 0 0 0 0 0


IR 11000111
PC 0000000000000010
SP 0000000000000000
Reg B 00000000
Reg C 00000000
Reg D 00000000
Reg E 00000000
Reg H 00000000
Reg L 00000000

ALU   0
ACC   00000000
TEMP  00000000
FLAGS: ZO, P, S, CY, AC 0 0 0 0 0


IR 00000001
PC 0000000000000011
SP 0000000000000000
Reg B 01101001
Reg C 11000011
Reg D 00000000
Reg E 00000000
Reg H 00000000
Reg L 00000000

ALU   0
ACC   00000000
TEMP  00000000
FLAGS: ZO, P, S, CY, AC 0 0 0 0 0
```

6.3 Synthesis and Optimization of the VHDL design

Synthesis is the process of transforming one representation in the design abstraction hierarchy to another representation. AUTOLOGIC is such a tool from Mentor Graphics, which is used to synthesize the compiled VHDL design into gate-level schematics and to optimize this description. AUTOLOGIC is used to optimize gate-level design for area by applying specified options. We chose the option to map the synthesized design to a technology like CMOSN, and to use algebraic and Boolean techniques to combine terms or eliminate redundant logic. The optimized top-sheet schematics shown in Figure 11 have five modules connected with each other, with each corresponding to a component of the VHDL design of the 8080. Each of the five modules in the top-Sheet consists of gate-level schematic sheets. One sample gate-level sheet is shown in Figure 12. There are about 90 such gate-level sheets in total for the whole synthesized design, and there are about 5400 logic gates involved overall. The synthesized schematic is also simulated to make sure that the synthesized design functions the same way as the validated VHDL model.

6.4 VHDL Modeling Summary

An abstract behavioral VHDL model was developed and verified. And then the behavior model was decomposed into the synthesizable VHDL model which was used to produce the gate level design of the 8080. There are about 4500 lines of VHDL code for the synthesizable version including the assembler which was developed also in VHDL and contains 800 lines of code.

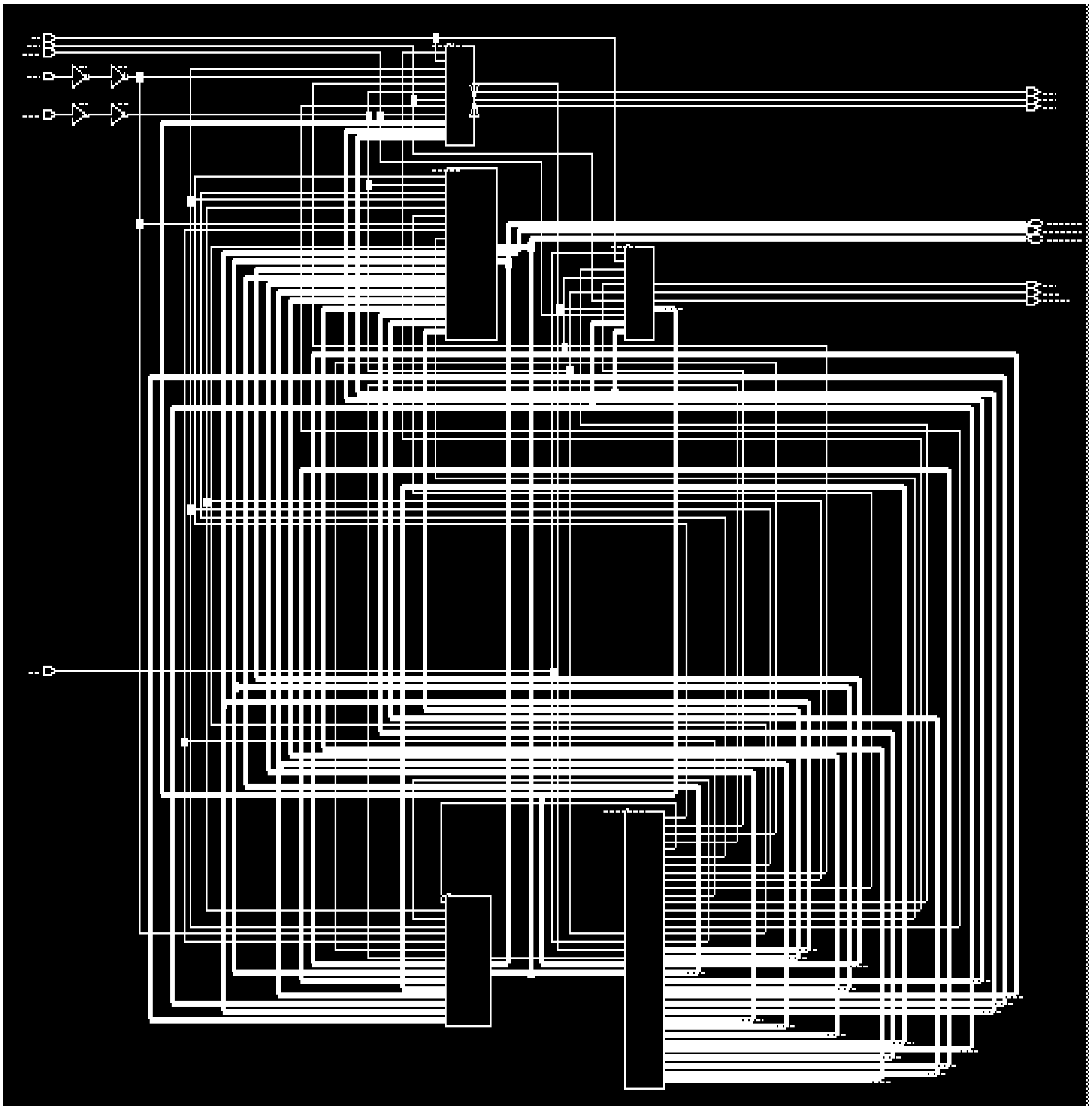

Figure 11: Top-Sheet of the Synthesized Schematics of 8080 CPU Model

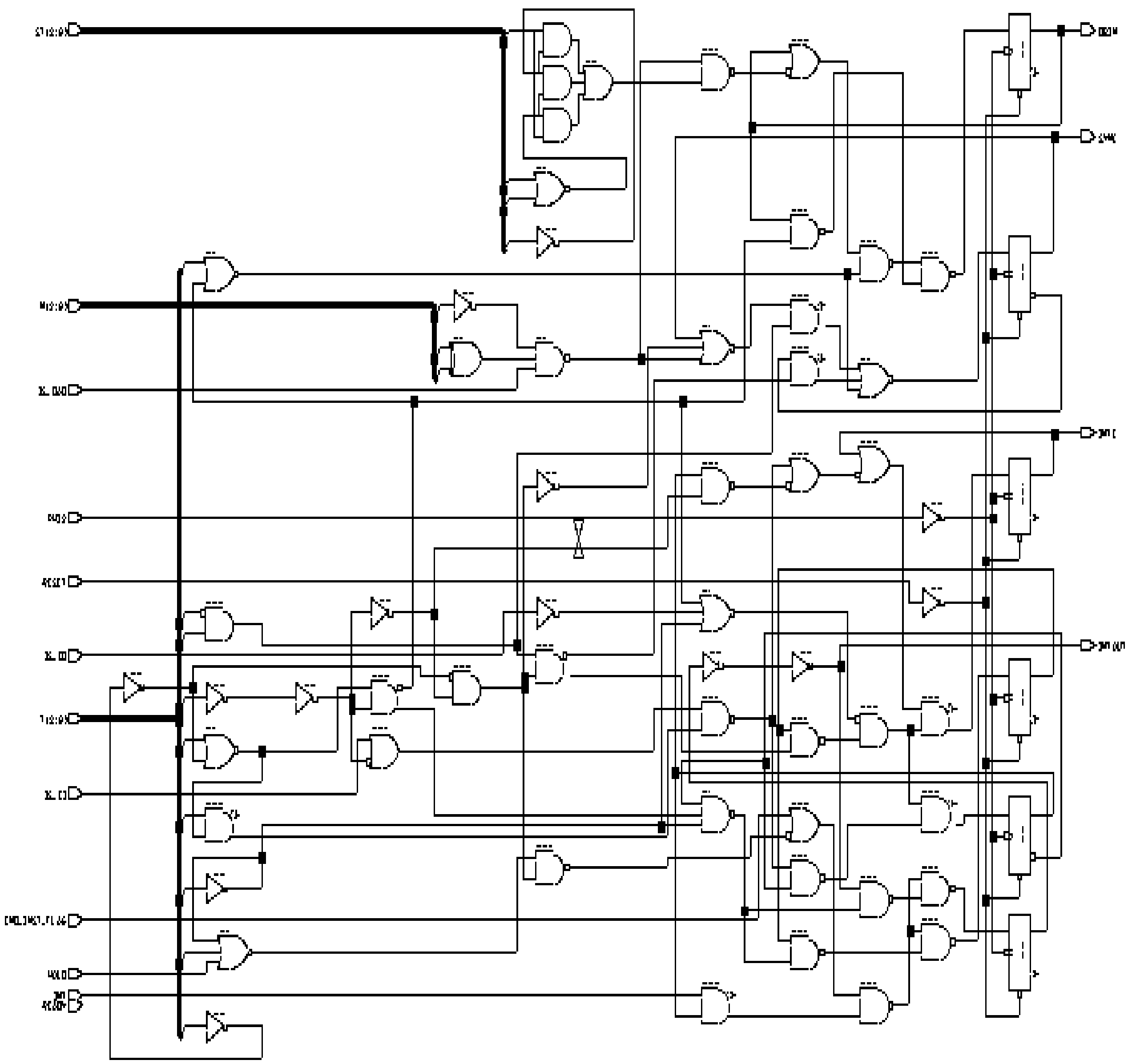

Figure 12: One Sample Gate-Level Sheet

References

[1] James R. Armstrong. *Chip-Level Modeling with VHDL.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[2] James R. Armstrong and F. Gail Gray. *Structured Logic Design with VHDL.* Prentice Hall, Englewood cliffs, New Jersey, 1993.

[3] Mentor Graphics. *Autologic VHDL Synthesis Guide.* unpublished work of Mentor Graphics Corporation, Mentor Graphics Corporation, 1993.

[4] Mentor Graphics. *QuickSim II User's Manual.* unpublished work of Mentor Graphics Corporation, Mentor Graphics Corporation, 1993.

[5] Mentor Graphics. *System-1076 Quick Reference.* unpublished work of Mentor Graphics Corporation, Mentor Graphics Corporation, 1993.

[6] M. Morris Mano. *COMPUTER SYSTEM ARCHITECTURE.* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[7] Intel marketing communications. *The 8080/8085 Microprocessor Book.* A Wiley-Interscience Publication, Intel Corporation, 1980.

[8] Cary Ussery Roger Lipsett, Carl Scharfer. *VHDL: Hardware Description and Design.* Kluwer Academic Publishers, Norwell, Massachusette, 1989.