TR-O-0117

05

DBFマルチビームアンテナディジタル信号処理部の開発（設計データ付き）

田中　豊久

1996. 3.15

ATR光電波通信研究所

目次

第１章　はじめに

昨今、世の中は移動通信ばやりである。電車の中、会議中、レストランで携帯端末の呼出音が鳴る。浸透の速度は目覚ましいものがある。人々は一度移動通信の便利さに気が付けばもう、元に戻れないかもしれない。この勢いで市場が需要が伸びれば将来的に周波数が欠乏状態になることは簡単に予測できる。この問題を解決するためには、新たな周波数の開拓、周波数の繰り返し利用、多重化技術の利用による回線の確保などが検討、研究されている。また、将来的には音声通信にとどまらず、画像通信、B-ISDNに代表される広帯域データ通信などの要求が高まってくるであろう。

一方、電子機器分野では、20年程度前からディジタル化の波が押し寄せて来ている。通信も例外ではなく、既にディジタル通信はサービスが開始されている。今後もディジタル化された通信は益々発展して行くものと予想される。

ATRでは設立当時から将来の高機能移動通信用アンテナとして、ディジタル信号処理を利用したDigital Beam Forming Antennaを提案、研究して来た。このアンテナはアンテナ部に円環パッチアンテナとリングアンテナを2層構造にして組み合わせたセルフダイプレクシングアンテナを用い、さらにそれをアクティブアレーアンテナ構成にしている。アナログ部も集積化が期待できるMMICによって構成も可能である。最後に複雑な信号処理を引き受けるディジタル部に並列処理ディジタル信号処理装置(DSP)で構成される、各種の技術の組み合わさった統合化アンテナである。当初ディジタル信号処理部は汎用DSPチップの搭載したボードを複数枚使用して構築されたDSPシステムを用いて研究が行われていた。汎用ボードを組み合わせたシステムでは、複雑な並列処理と相互のデータのやり取りを制御する事や動作速度の限界があり、また汎用システムであるがゆえに筐体も非常に大型にならざるを得なかった。ここで、より小型化、複雑な処理の実現を目指したDSP部のASIC化に白羽の矢が立った。最終的な目標はすべてのDBFアンテナの処理が1チップに集積化されたASICの完成であるが、1チップASICは作成にコストと汎用性を犠牲にする事を要求する。そこで設計がASICと同じ手法が用いられ、設計自体もASICに転用可能なFPGA(Field Programmable Gate Array)を用いたDSPを構築する事になった。FPGAを用いる利点として、ユーザサイトで設計開発が可能な事、ASIC化へのハード規模などの情報が得られる事が挙げられる。逆に、欠点としてはそのプログラミングが可能なアーキテクチャにより動作速度がASICと比べて劣っている。しかしながら実験システムではデータレートを低く抑える事により、その欠点をカバーすることができる。このようなATRでの歴史的背景のなか、FPGAを用いたDSPボードにより、DBFアンテナの実現性を検証する事を目的としたDSP開発が行われた。本研究レポートでは学術論文等では、取り扱わなかった、設計のノウハウを含めてDBFアンテナが作れるようにまた、既存のシステムが理解できるように記述した。DBF技術を用いたFFTマルチビーム生成部、ビームセレクタ部、位相補正部までを本テクニカルレポートに、CMA(Constant modulus Algorithm)を指導原理としたアダプティブプロセッサについてを参考文献[1-1]に、ビームスペースでアレーアンテナで最大比合成を行なうDBFセルフビームステアリング用プロセッサについては参考文献[1-2]で述べる。

第２章　Digital Beamforming Antenna

DBFアンテナはアレーアンテナ構成をとり、アンテナの各素子で受信された信号をディジタル信号処理する事により任意の方向にビームを形成するアンテナである[2-1]。

従来、アレーアンテナの制御方法としてはアナログ式のフェイズドアレーアンテナがある。フェイズドアレーアンテナはフェイズシフタを用いて所望のパターンを生成するものであるが、フェイズシフタが高価なマイクロ波デバイスの為、通信用としてはあまり普及していない。またアナログでのマルチビーム生成には、バトラーマトリックスなどがあるがアンテナの素子数が多くなるに従い回路規模が増大し移動体通信には適応しにくくなる。一方、DBFアンテナではアナログのフェイズシフタをディジタルデバイスに置き換える事と等価であるので、より小型で安価な構成が期待でき、さらに受信信号のレベルを劣化させる事なくマルチビーム生成が可能である。またディジタル信号処理と親和性のよいアダプティブアルゴリズム[2-2]-[2-8]で動作させれば不要な干渉波の除去も可能であり、マルチビーム形成はこのプリプロセッサとしても機能する。リアルタイムに生成されるマルチビームの最も大きな振幅を持つビームを選択すれば、所望波の自動追尾の機能も得られる。

以上の特徴により、DBFアンテナは移動体通信における基地局及び移動局の出力の省電力化や周波数の有効利用などに期待されている。既にDBFアンテナについてはワークステーションを用いたオフライン処理での試作や、汎用DSPを用いた試作が報告されているが、DBFアンテナを高速かつ高機能に実現するためにはディジタル信号処理部のASIC (Application Specific Integrated Circuit)化が不可避である。

第１節 DBFアンテナシステムの概要

DBFアンテナシステムは表1.に示す諸元で試作した。図2-1に示すアンテナ部は16素子のパッチアンテナからなり、FFTアルゴリズムを用いるため、4×4の四角形に配置し、隣接素子アンテナの間隔は$\lambda/2$ (λ:搬送波の波長)とした。搬送波周波数はL帯(1.545GHz)であり、想定したデータレートは16kbpsである。システムとしては、アンテナからA/Dコンバータまでのアナログ部、A/Dコンバータ以降のディジタル部に分けられる。

表2-1. ATR開発システム概要

ITEM	SPECIFICATION
搬送波周波数	*1.545GHz (L帯)*
アンテナ素子数	*16素子（４×４）*
IF周波数	*32kHz*
A/Dコンバータ サンプリング周波数	*128kHz*
データレート	*16kbps*
変調方式	*π/4シフトQPSK*

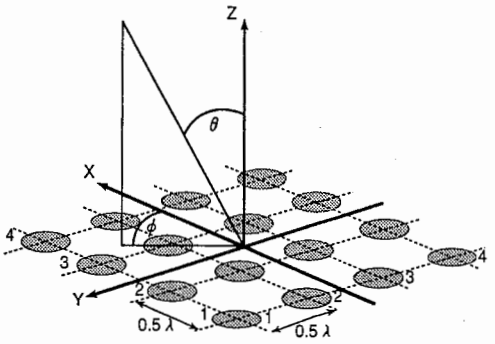


図2-1. アレーアンテナの配置図と座標系

第２節 アナログ部概要

図2-2.にDBFアンテナのブロック図を示す。アナログ部はアンテナ部、周波数変換部、フィルター部、A/Dコンバータ部からなる。各アンテナ素子ごとに受信された信号は周波数変換部(D/C)で、1.545GHzから32kHzのIF信号に変換され、続くフィルタ部で高調波が除去される。次にIF信号はサンプリング周波数128kHzでA/Dコンバータにより8bitのディジタル信号に変換された後、ディジタル信号処理部に渡される。ディジタル信号処理部に関する詳細は以降の章で機能毎に述べる事とする。

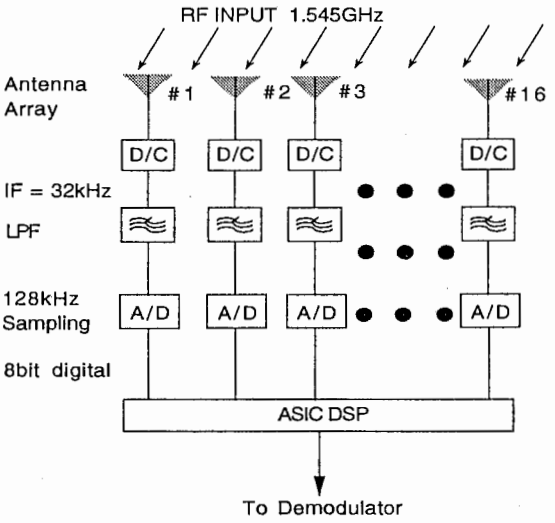


図2-2. DBFアンテナ構成図

第 3 章　Front-end Board

第 1 節　Specification

Front-end boardはDBFアンテナのアナログ-ディジタルを結ぶパイプ的な役割を果す。ボード上には送信用に16素子に対応する16個のD/Aコンバータと受信用の16素子からの入力に対応する16個のA/Dコンバータが搭載されている。表3-1にFront-end boardの仕様を示す。

表3-1. FRONT END ボード仕様

Front End ボード共通仕様		
ITEM	Specification	Condition
ボード	VXI C-size ボードに準拠	
入力	Master Clock	
	Analog 16ch	To A/D Converter from D/C
	Digital 16ch/12bit	To D/A Converter from 変調DSP
	A/D, D/A Enable/Disable	To D/A, A/D from 復調/変調DSP
出力	Analog 1 6ch	From D/A Converter to U/C
	Digital 16ch/12bit	From A/D Converter to 復調DSP
GND	Digital/Analog Separated	Connected at Back plane
A/D Coverter部の仕様		
ITEM	Specification	Condition
Sampling Rate	Max 256KHz	
INPUT	16 ch. of element data	
INPUT Impedance	50Ω nominal	
Input Level	+/-5V	A/D : BB ADS7800KP
OUPUT	16ch/12bit Digital	192pin + α (DGND、Control)が必要
Digital出力形式	12bit 並列	
OUTPUT Impedance	68Ω Nominal	
Output Level	CMOS	
Timing Synchronization	20nsec	16ch. のSampling ClockのSkew
DC Accuracy	0.5%	
SNR	65dB	fin:47KHz
DC Linearity	+/-2LSB	
Power	+5V	+VS
	-12V	-VS
D/A Converter部の仕様		
ITEM	Specification	Condition
Conversion Rate	Max 256KHz	
INPUT	16 ch/12bit digital data	192pin + α (DGND、Control)が必要
INPUT Impedance	68Ω nominal	
Input Level	CMOS	
Digital入力形式	12bit並列	D/A Converterの下位2bitはdisable
OUPUT	16ch of element signal	
OUTPUT Impedance	50Ω Nominal	
Output Level	6Vp-p	D/A : Aanalog Device AD7840
Timing Synchronization	20nsec	16ch. のConversion ClockのSkew
DC Accuracy	0.50%	
THD/SNR	72dB	Vout=1KHz, fconv=100KHz
DC Linearity	+/-1LSB	12bit full scale
Power	+5V	+VS
	-5.2V	-VS

A/Dコンバータにはバーブラウン社製のADS7800を、D/Aコンバータにはアナログデバイス社製のAD7840を採用している。[3-1],[3-2]

第1項　入出力電圧範囲

A/Dコンバータは入力電圧として最大+/-5Vをサポートしている。この場合12bit精度フルスケールで最大値をとるため、8bit精度でDSP部が動作する時入力電圧範囲を限定して使用する。つまり上位3bitと最下位1bitを使用しない場合、Vpp=1.25V入力がフルスケールとなる。これはDSPボードの設計により変更が可能である。また、採用したA/Dコンバータのディジタルデータフォーマットは2の補数形式ではなく、オフセットバイナリー形式であったのでDSPボードで演算前に2の補数形式に変換する必要があった。

第2項　Sampling timing

FRONT END ボードのA/D、D/AコンバータにはDSPボードからサンプリングのタイミングを決めるClockが送られる。開発システムでは128kHzのClockが用いられている。

第2節 構成

図3-1にFront Endボードの外観図を示す。上段が送信用D/Aコンバータで下段が受信用A/Dコンバータである。フロントパネル側に4つのハイピッチ120pinコネクタがあり、上下2つずつ組みで受信用DSPボード、送信用DSPボードと接続される。

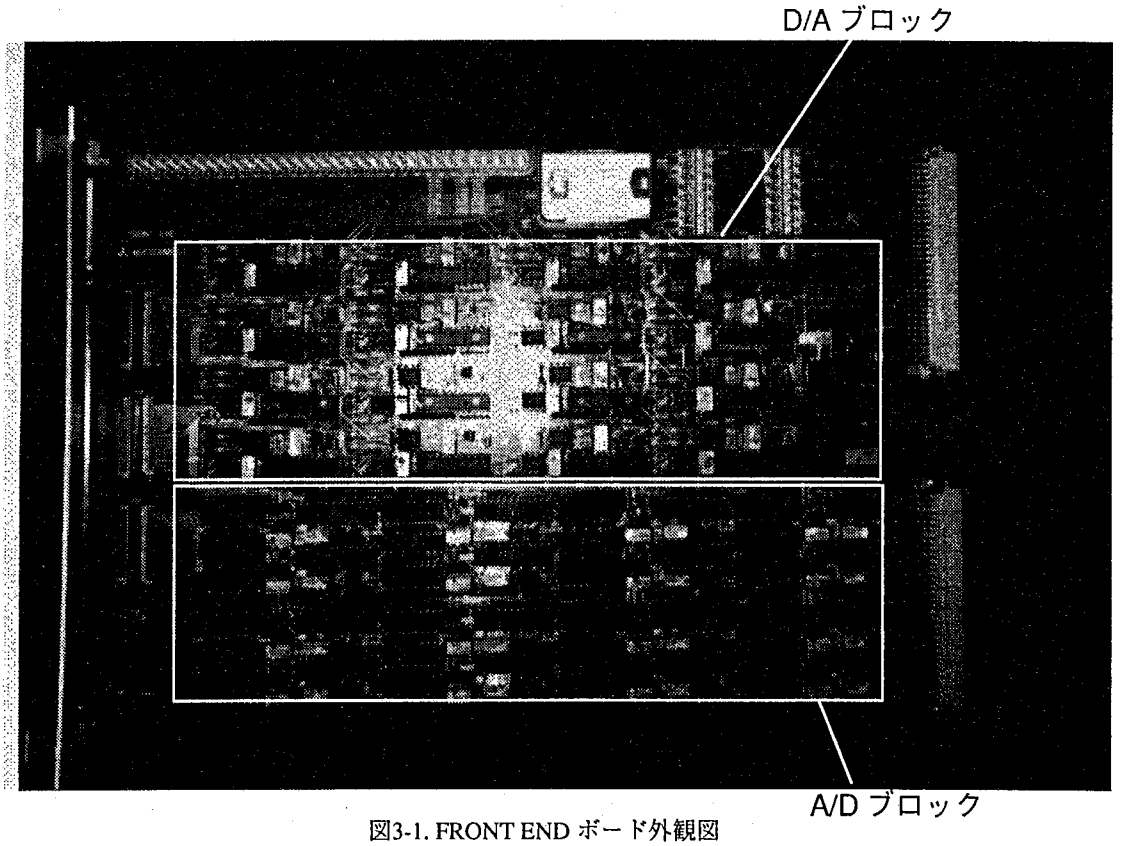


図3-1. FRONT END ボード外観図

第 4 章　Digital Signal Processor Board

第 1 節　Specification

第 1 項　DSPボードの仕様を表4-1に示す。DSP Boardは6層基板で設計されており。電源層(1層)，GND層(1層)，パターン層(4層)からなる。試作DSPボード上に搭載されている10個のFPGAにはu1からu10の部品番号を付ける。DSPボードの外観図を図4-1に、部品番号を示したイラストレーションを図4-2に示す。

表4-1. DSPボード 仕様

ITEM	Specification	Condition
Board Size	34.4cm x 23.3cm	VXI C-size
Number of FPGAs	10	PGA223 pin用socket付き
INPUT Connectors	Master Clock	J7(PO6 @Front pannel)
	Sampling clock OUT (u10 - E18)	J8(PO6 @Front pannel), J3
	Reserve (Connected to u1-T4 pin)	J9(PC6 @Front pannel)
	Digital 16ch/12bit	J3, J4 From A/D Converter (120pin)
	A/D Enable/Disable	J4 To A/D Converters
	A/D Status	J3 From A/D Converters
	FPGA down load	J9 (18pin)
	CPU I/F	J1,2 (96pin)
OUTPUT Connectors	Digital 24bit + 2 clock	J5 (34pin)
GND	Digital/Analog Separated	Connected at Back plane
Master Clock Frequency	7.04MHz	0V cross +/- 4V MAX, Pulse
Input Level	CMOS	
Output Level	CMOS	
CPU I/F	Register based A24/D16	

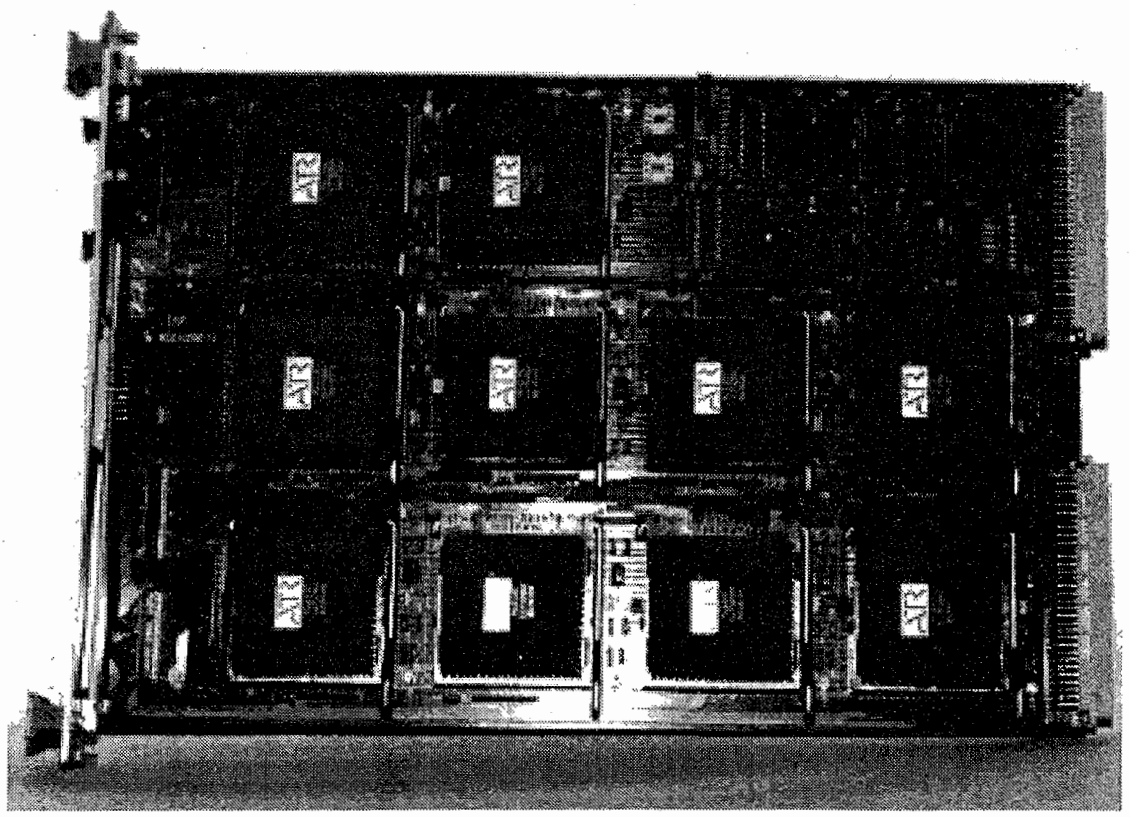


図4-1. DSP ボード外観図

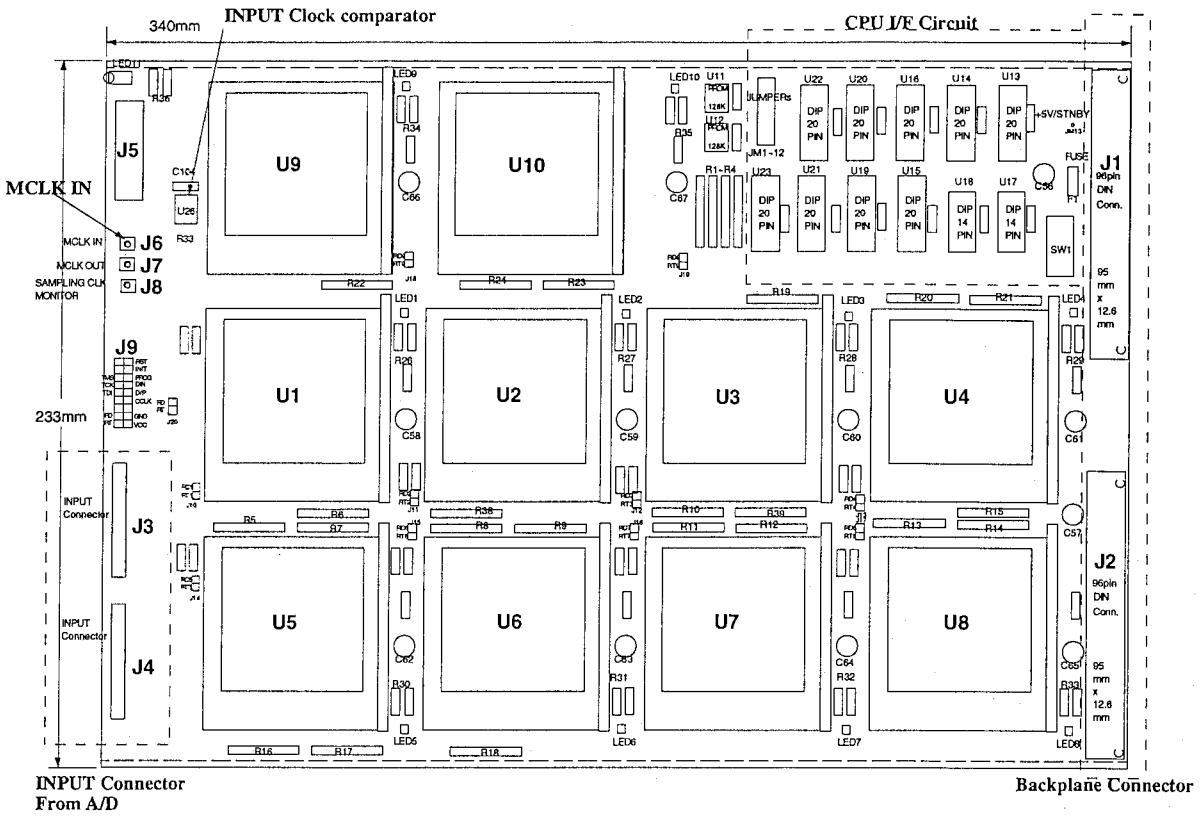


図4-2. DSP ボードイラストレーション

第2節　VXIbus

Front-end boardとDSP boardに電源と通信プロトコルを供給しているのがVXIbus規格のフレームである。VXIとはVMEbus Extensions for Instrumentsの略で文字通りVME Busを計測器の使用に耐えうるように拡張したものである。VMEにはなかったEMIと冷却についての仕様を追加し、さらに通信プロトコルの仕様も拡張している。

第1項　CPU I/F(通信プロトコル)

試作したDSP BoardはVXIの通信機能を備えており測定データの読み込み，パラメータの設定が可能である。

1. Register basedとMessaged based通信

VXI規格では通信プロトコルとして

1. Message Based
2. Register Based

の2種類がサポートされている。Message based通信はコマンダあるいはコマンド・ベースド・サーバント機能と呼ばれる通信用装置をコマンダ(Computer側)、サーバント(Module側=DSPボード等の機器)共に備え、より高度な通信が可能となっている。Register based通信はサーバントが簡単な回路構成により通信が行える特徴を持っている。DSP BoardはRegister Baseの通信をサポートしている。以下ではRegister basedの通信方式

について述べる。Messaged basedについては参考文献[4-2]を参照されたい。

2.Register basedプロトコル

Register Baseの通信では，A16，A24，A32の3つのAddressing modeがサポートされている。1デバイスあたり、A16では64bytes，A24では最大4MBytes，A32では最大4GBytesまでで、空間全体の1/2を限度に割当可能となっている。

A16 mode
Register based通信で必ず必要なモードである。u10, u9のデータ読み書きはA16モードで行われる。

A24 mode
A16 modeをもち、さらにA24モードに拡張できる。u1～u8はA24での通信によりデータの読み書きができる。

Read/Write Timing
Register basedのCPU I/F回路を構築するためには、I/FのRead/Write Timing仕様を満たすように設計する必要がある。DSPボード内ではRead/Write用パルスを発生させるため、VHDLで記述したシーケンサ(付録 VHDLソースコード"ss_cont"参照)を用いている、実際の動作はVHDLの記述で確認できる。このシーケンサはVXIから供給されている16MHzのクロック(SYSCLK)に同期して動作するように設計した。Timing的には若干余裕を見て設計している。図4-3、4-4にCPU I/FのTiming Chartを示す。

Read timing chart

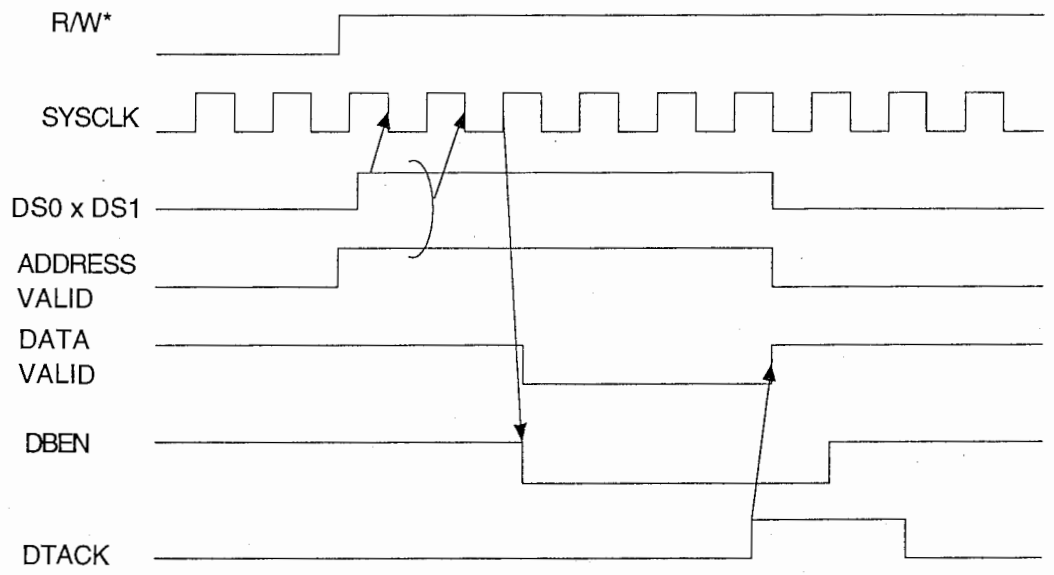


図4-3. Read timing

Write timing

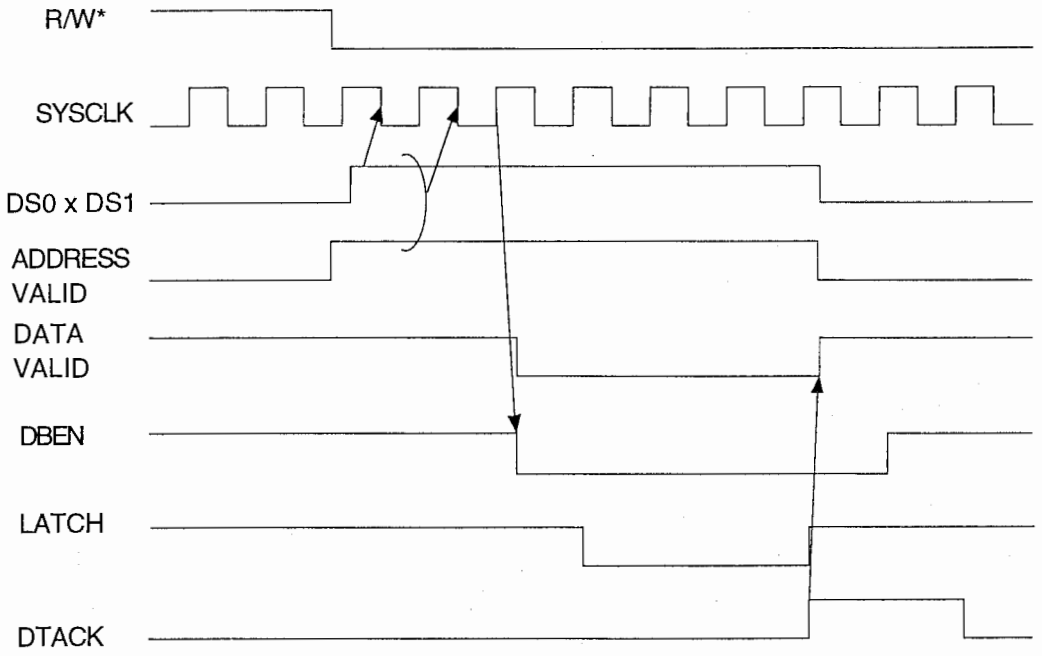


図4-4. Write timing

Sample program

読み書き用の簡単なサンプルプログラムを付録に添付しておく。

開発システムのVXIのコントロールはHP社製のVXIのフレーム内で使用できるWorkstationを使用した。Workstationのオペレーティングシステムは HPUX9.05で、VXIのコントロール用はSICL(Standard Instrument Control Library)というC言語で用いられるライブラリを利用して行なった。簡単な読み書きのSample Programを付録に添付しておく。また、VXI BUS回りのDSPボードの構成を図4-5に示す。図では、バックプレーンのBUSに直接繋がる汎用ロジックICを通して各信号がFPGAに渡されている。

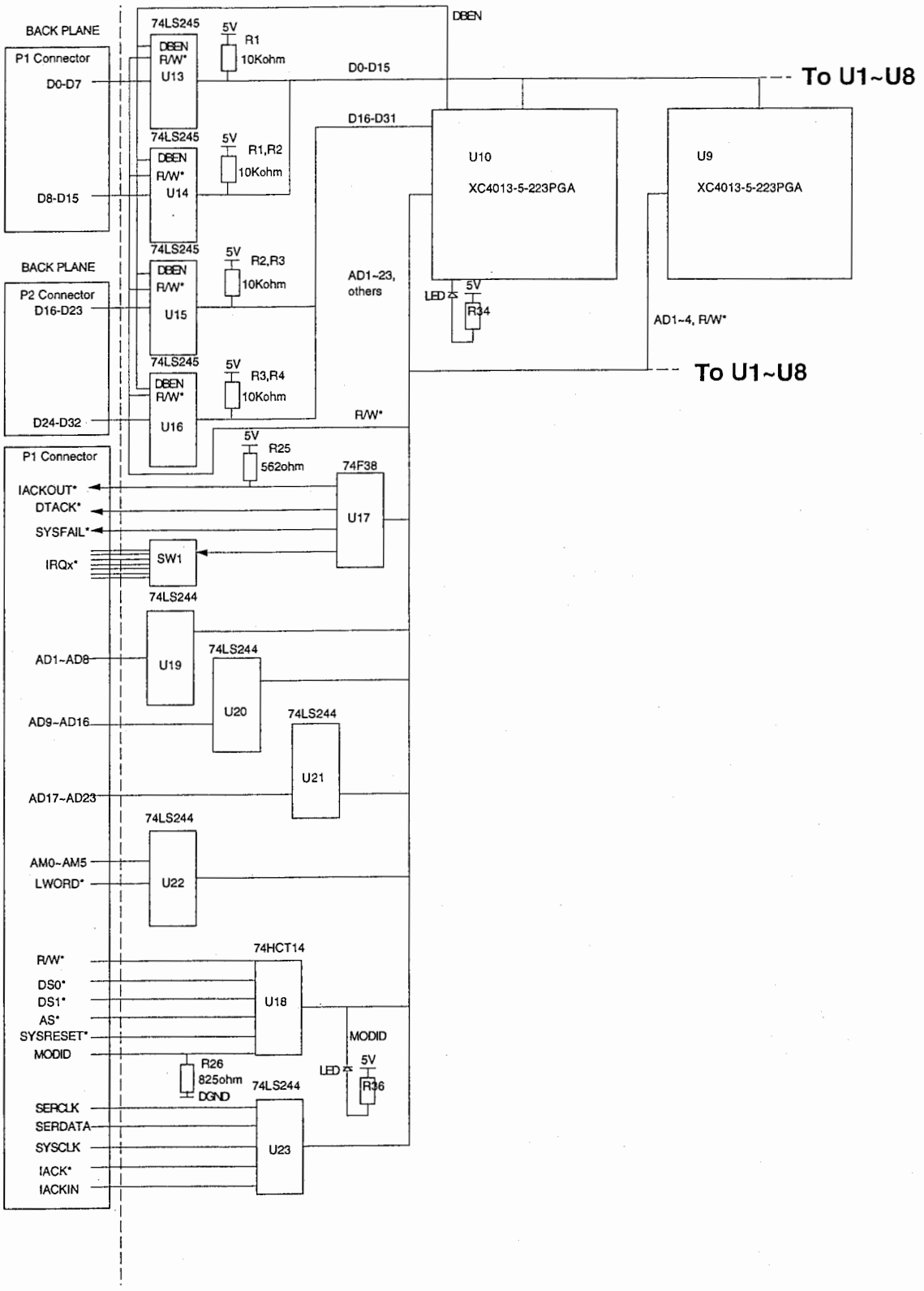


図4-5. DSPボードのCPU I/F回りの構成

第3節　FPGAへのconfiguration data down load について

FPGAへ設計データを読み込ませるためには次の2種類の方法を想定して設計してある。

1. フロントパネル側のコネクタJ9にXilinx社のxchecker cableを接続し、PCあるいはWorkstationから10個すべてのFPGAにダウンロードする方法。
2. Serial PROMにu10のbit fileを焼き付けておき，ボードへの電源投入時にu10のみを立ち上げ，CPU I/F経由で残りの9個を立ちあげる方法。

通常、u10のデザインは頻繁に変更されるので、実質的には2の方法をとる。1と2の方式の選択はDSPボード上のJumperを切り替えで行なう。FPGA configuration回りの詳細接続を図4-6、図4-7に示す。第4節　Clock system

第1項　Clockの流れ

Front-end BoardおよびDSP Boardのクロックは DSP Boardに供給されているMaster Clockを源泉として分周されて用いられている。Master clockの仕様を表xxに示す。Function generatorなどから発生されたClockはフロントパネルのコネクタから入力され、コンパレータで受け0V クロスのアナログ信号をTTLレベルの出力に変換してu10に渡される。u10はすべてのFPGAのクロックを管理している。また、DSPボード内部のGND系から分離するため入力側は別GNDになっている。クロック系の流れを図4-8に示す。

第2項　DSP system同期の考え方

1. start_clk
u10のクロック管理部では1イタレーションのタイミングを決めるスタートパルスを発生する。このパルスがすべての演算の始まりとなる。
2. MCLK
DSPボードのシステムは単層完全同期システムで実現されている。すべてのプロセッサはu10から供給される同期したMCLK(7.04MHz)を用いており、一元管理されている。
3. Phase CLK
これのクロックは4サンプル(= 4 start_clk)に1回出力される。

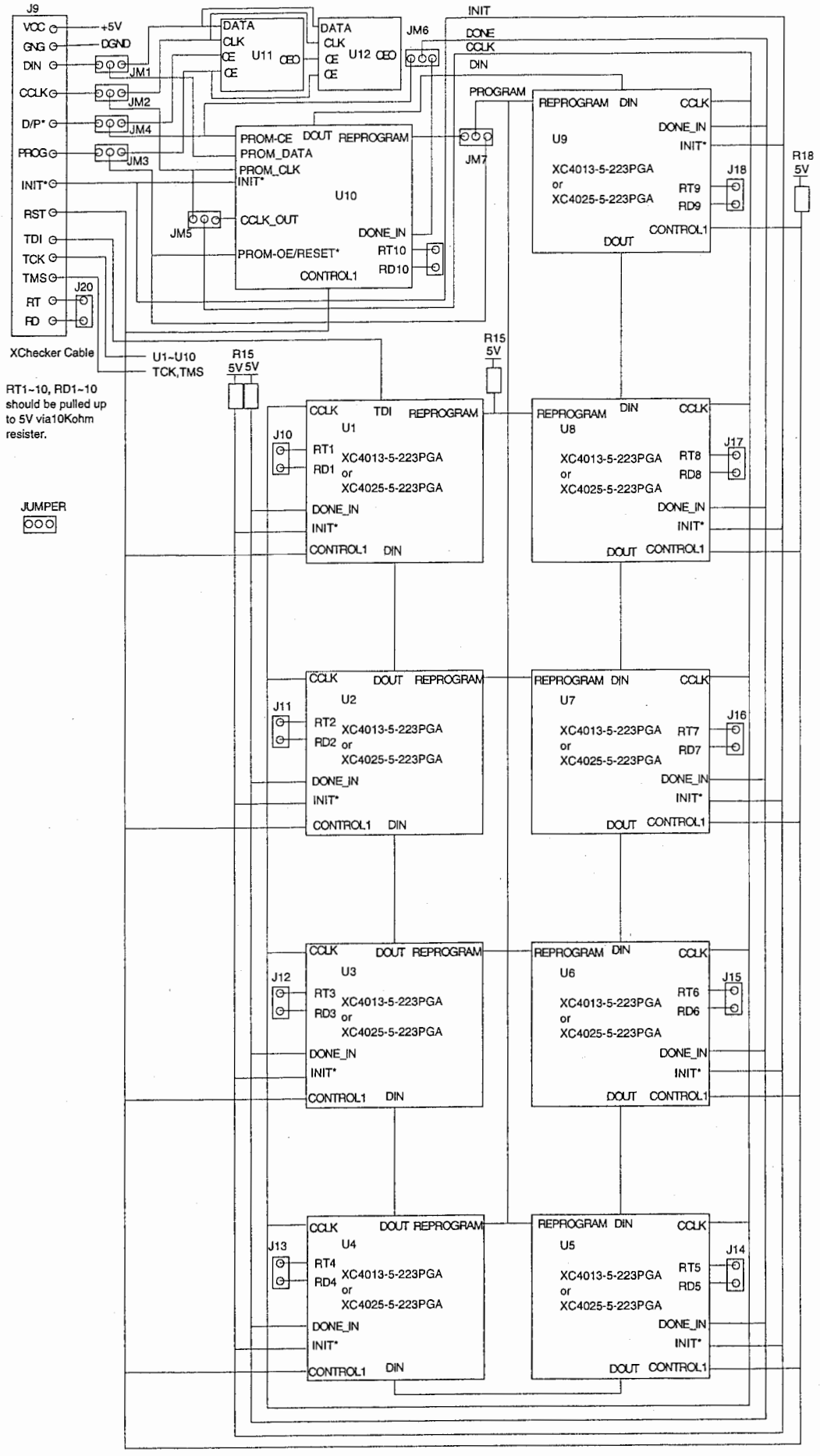


図4-6. DSP ボードでのコンフィグレーション用信号の接続

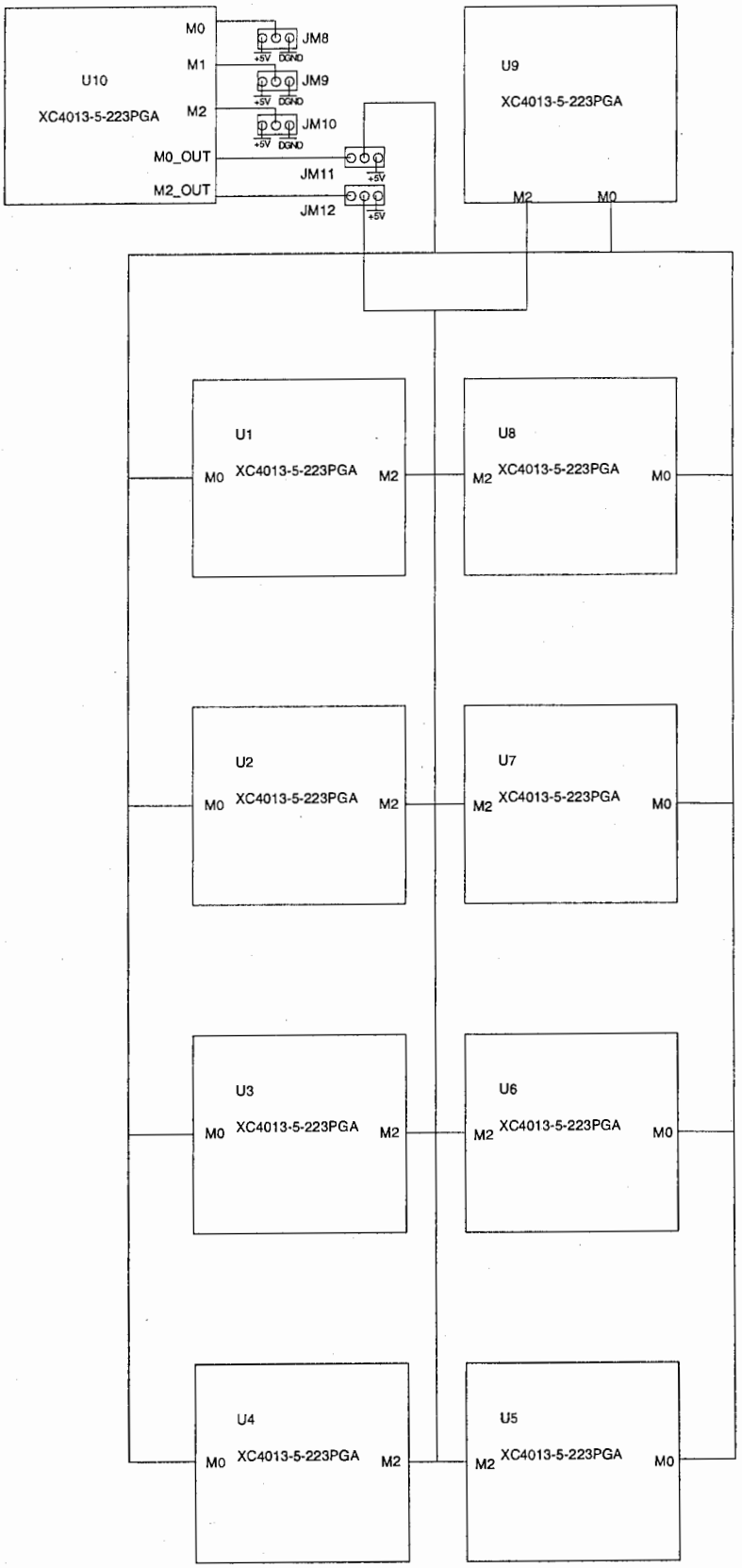


図4-7 DSP ボード コンフィグレーションモードの設定

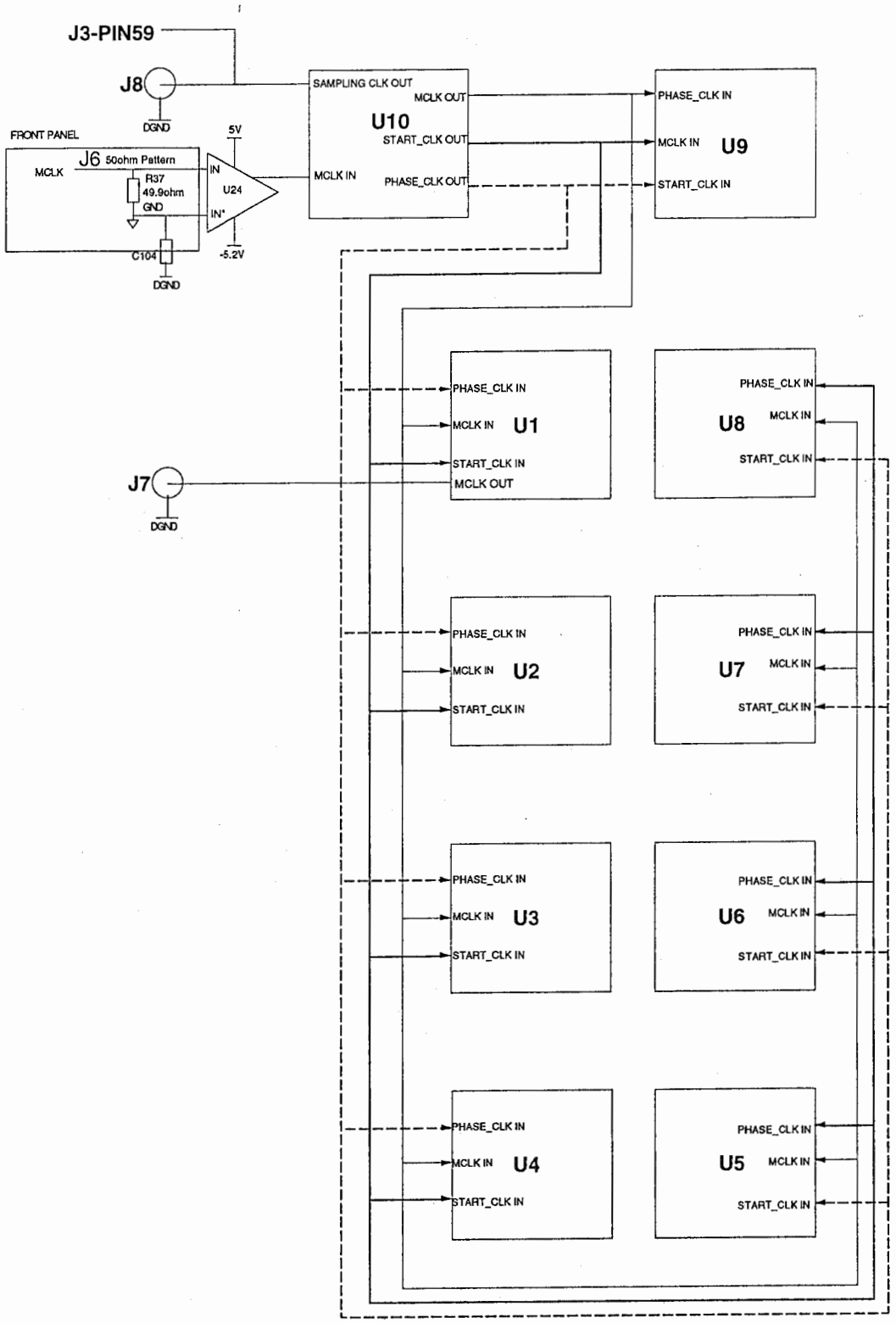


図4-8. DSP ボードでのクロックの流れ

第５章　FPGAとその開発

第１節　ASIC選定

まず、開発にあたり採用するデバイスの選定を行った。デバイスの種類としては、ユーザ側で開発可能なデバイスとしてFPGA(Field Programmable Gate Array)を採用する事になっていた。FPGAを用いてDSP部を開発する事は最終的な目標形態である"小型化"=1chip ASICと汎用DSP chipとの妥協点であり、開発期間と開発のリスクの面からもメリットがあった。さてFPGAには採用されたテクノロジーにより主に次の2種類に分けられる。

1. Antifuse type(Sea of gate type)
2. SRAM type

1のAntifuse typeはデバイス内に敷き詰められたゲート間のAntifuseを切る(すなわち結線する)事によってプログラミングされる書き込みが1回限りのデバイスである。特徴としてはマスクタイプのゲートアレーに構造が似ていて，移植性がよい事が期待できる。また動作速度も比較的高速にできる。
一方，ATRで採用したSRAM TypeのFPGAの最も大きな特徴は，何度でも書き換えが可能な点にある。デバイス内部の構成は回路を構築できる論理回路が島のように配線領域の中に点在している。比較的大規模な機能を構築するためには複数の論理ブロックを用いなければならず，それらの間を配線するので遅延が大きく動作速度が上がらないという特徴もあわせもつ。
DBFアンテナからの要求仕様は最低でも128 kHzサンプリングで16素子の入力信号を処理出来なければならない事であったので，FPGAは高速で可能な限り大規模のものが要求された。
ここで，どちらのタイプを選定するかという決定をしなければならないときに物の入手性にも影響される。当初はCrosspoint社の1万ゲート規模のデバイス(1のタイプ)とXilinx社の1万3000ゲートのデバイス(2のタイプ)を候補に挙げた[5-1]。結局，その時点でテクノロジー的に未熟であったCrosspoint社のデバイスは入手不可能になり，Xilinx社のXC4013というデバイスを採用する事になった。

第２節　FPGAと開発ツール

デバイス選定で開発ツールとの相性，充実度もファクタとなる。回路設計ツールのみならず，FPGAであるがゆえの配置配線ツールが重要となる。回路設計や論理合成などを含めCAD自体がCPUパワーを必要とするアプリケーションは配置配線用ソフトウェアである。配置配線ツールによって設計のスループットに影響し、また配置配線できる回路規模も違ってくる。また回路の動作周波数もこの配置配線により決まってくる。例えば旧Neocad社(現Xilinx社)のFPGA Fundaryという配置配線ソフトでは経験的に、13,000ゲートのもので4000コネクション、25,000ゲートのもので6,000コネクション程度が限界である事がわかった。それらの値を越えた場合でも配置配線できるが、時間が飛躍的にかかってしまう。

第３節　FPGAの現状と展望

1996年2月現在のATRの使用デバイスはゲート数25,000のXC4025である。今後もFPGAはASICの機能検証用として用いられるであろう。数年内に10万ゲート規模のデバイスが実質的に入手可能になる。(FPGAメーカの新製品発表はあてにならない。発表後6ヶ月から1年先に入手できると見積もった方が無難。) 近未来において有望なFPGAを挙げておく。

1. ORCA (ATT)

Xilinx社のFPGAのセカンドソースとして出発。SRAM方式。Xilinxのデバイスと比べ"島"が大きく大規模化とともに問題となってくる配線領域への負担が軽い。また現在4万ゲートまで入手可能。

2. ALTERA

SRAM方式。ラインナップに10万ゲートを詠っているが，入手は数年後か。FPGA選定当時ネックとなった，RAMを構築できない特徴は最新モデルでは改善されたらしい。

3. Xilinx

今後も老舗として実績を伸ばして来るであろう。最近の傾向として大規模デバイスには重きをおいていないようだ。

第4節　開発ツール

開発ツールはいつも、そのソフトウェアの完成度との騙し合いになる。やはり高価なCADツールといえどもバグは必ず付き物である。限られた時間内に成果を出す(納期に間に合わせる？）ためには、例えソフトに不具合があったとしても、それを何とか避けて開発を続けなければならないのが現状である。新たなバージョンのソフトウェアが来ても、それにしなければ無い理由が無く，特に急ぎの開発が必要な時には、現状の設計ツールが問題なく動作しているならバージョンアップは行なわない方が賢明である。本節ではDSP部開発に使用した開発ツールについて述べる。設計データ付きのテクニカルレポートで、実際にデータを使用する場合にはMentor Graphics社のDesign Architectが必要である。またXilinx社のXACT5.0以降でMentor Graphics社対応の開発用ソフトウェアが必要である

第1項　デザインツール

回路設計のツールにはMentor Graphics社のEDA(Electric Design Automation)でFalcon Frameworkというソフトウェアを使用した。さて、最近言語にてハードウェアの仕様を記述し、それをCADソフトウェアによって実際の回路に変換する、論理合成ツールが実用化されてきた。回路設計の方法としてはコンベンショナルな回路図入力によるものと、先程述べた言語を用いた方法がある。Mentor Graphics社のツールでは回路図入力、記述言語(VHDL: Very high speed IC Hardware Description Language)の両方をサポートしている。VHDLという記述言語はもともと米国の国防総省で開発され、IEEE-1076という規格になり、ディジタル設計の分野では主流となっているものである。しかし、その言語を用いた設計では仕様記述のみで回路が構築でき、また上位層からの設計(Top downによる設計)が可能になるとうたわれているが、実際は各部品を言語で記述し、それらの結線情報をまた言語で記述するというものであり、完全ではない。また、記述言語から生成された合成回路はFPGAの中に割り付ける際、配置配線の効率が良くないとういう欠点がある。従来の回路図で設計する場合では、FPGA設計ツールから提供されるライブラリと呼ばれる部品を用いて回路を構築する。この時の利点としては、比較的大規模な部品(例えば16bit Accumulatorなど)はFPGAに特化してオプティマイズがなされている事である。これは配置配線に有利で、つまりゲートを有効に利用でき、動作速度も速い。欠点としては、ライブラリで供給できないもの、組み合わせ回路やステートマシーン(シーケンサ)の設計がやっかいである事である。そこで、DSPの設計にはこれらの利点を組み合わせ、必要な機能をもつライブラリは極力利用し、複雑な処理を回路で構築しなければならないコントローラなどは記述言語で開発するという手法を用いた。回路図がすべて完成すると、そこからネットリストを抽出して、配置配線ツールに渡す。

第2項　配置配線ツール

ASICやGate Arrayなどでは、ASICベンダーでの仕事になるが、FPGAの場合配置配線まで行なわなければならない。配置配線ツールとしてはXilinx社とNeocad社のツールを使用した。Neocad社のツールはFPGAの設計ツール関連で汎用配置配線用ツールである。数種類のFPGAの配置配線のみを行なうがアルゴリズムが優れており、Xilinx社製のツールより有効であった。1995年にそのXilinx社に買収されてXilinxのソフトになった。

第3項 Debugging

回路設計における、検証方法として次の2つが挙げられる。

1 Simulator

回路設計段階での検証である。VHDLで記述した論理を確認する事や、回路図を用いてさらに上の階層で検証を行なう。

2 Logic Analyzer

FPGAに書き換えのものを使っているので、クリティカルなタイミング検証をSimulationで行なう方法と実際に回路に落とし、Logic Analyzerを用いて行なう方法がある。前者は可能であるが大規模になれば非常に時間がかかってしまう傾向がある。その場合、実際にロジックを実回路上で検証する方が有効な場合がある。

第 6 章　FFT Multibeam Forming ASIC Implementation

第 1 節　DBF Multibeam Antenna system Overview[6-1]

第 1 項　FFT Multibeam Operation priciple

図6-1.にディジタル信号処理部の概要を示す。

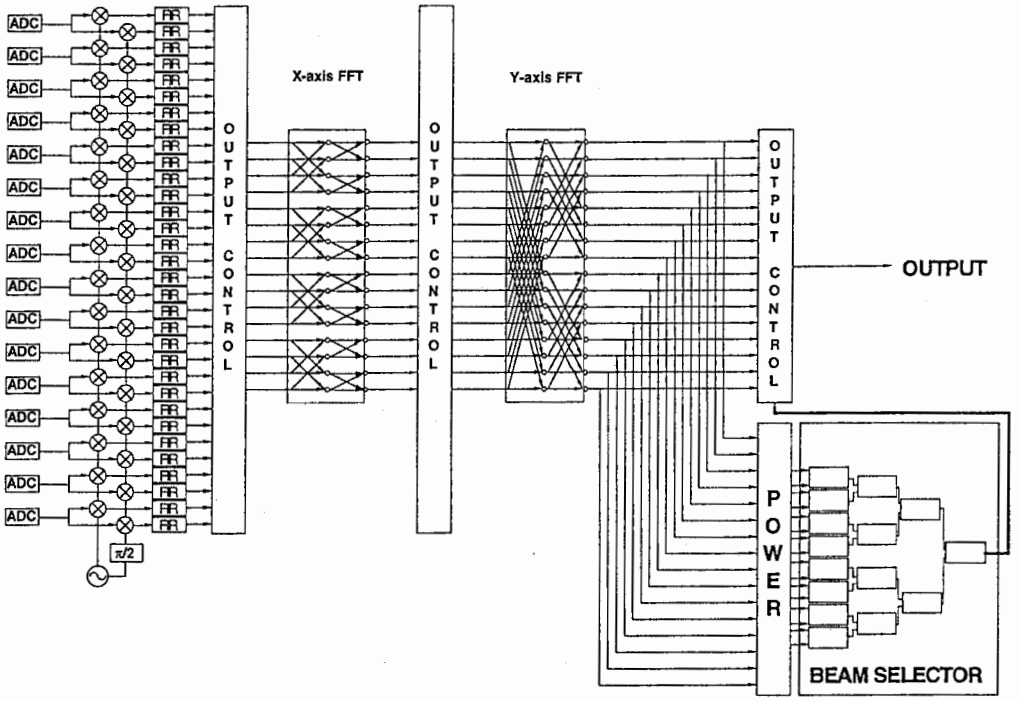


図6-1 ディジタル信号処理部の概要図

A/Dコンバータでディジタルデータに変換された信号は、マルチビームを生成する為に次の手順で処理される。

まず、素子アンテナの受信信号 S_{xy} は、

$$S_{xy}(t_l)=\cos(\omega_0 t_l+\phi_m+\theta_{xy}) \qquad (6\text{-}1)$$

x=1,2,3,4 y=1,2,3,4

と表わされる。ここでx,yは図3-1.に示す素子アンテナの座標を表わし、ω_0はIF信号の角周波数、t_lはサンプル時間を表わす。ϕ_mはQPSKによるディジタル変調位相である。θ_{xy}は各アンテナ素子における受信位相の共通のローカル信号からのずれを表わす。入力データは式(6-2)で表わされる準同期検波が行われ、I、Q成分に分けられる。ローカル信号は4サンプルを1周期とする信号で、0、$\pi/2$、π、$3\pi/2$のくり返しである。

$$\begin{bmatrix} i(x,y) \\ q(x,y) \end{bmatrix}=\begin{bmatrix} \cos(n) & 0 \\ 0 & -\sin(n) \end{bmatrix}\begin{bmatrix} S_{xy}(t_l) \\ S_{xy}(t_l) \end{bmatrix} \qquad (6\text{-}2)$$

x=1,2,3,4　y=1,2,3,4　n=0, $\pi/2$, π, $3\pi/2$.....

続いて準同期検波時に生成される高調波の除去とナイキストフィルタの機能を持つ10タップ50%ルートロールオフのFIRフィルタとして式(3)に示される演算がなされる。この演算は現在と過去9個のI,Qデータにフィルタ係数を掛ける積和演算である。

$$\begin{bmatrix} i_f(x,y) \\ q_f(x,y) \end{bmatrix} = \sum_{n=0}^{9} \begin{bmatrix} h(n) & 0 \\ 0 & h(n) \end{bmatrix} \begin{bmatrix} i_{k-n}(x,y) \\ q_{k-n}(x,y) \end{bmatrix} \qquad (3)$$

x=1,2,3,4　y=1,2,3,4

i_f及びq_fはそれぞれフィルタリングされたベースバンド信号である。$i_k(x,y)$、$q_k(x,y)$はそれぞれ現在のサンプル時のデータを表わす。次にマルチビームを生成するため、式(6-4)で示すX軸に沿ったFFTの計算を行う。

$$\begin{bmatrix} i'(x,y) \\ q'(x,y) \end{bmatrix} = \sum_{n=0}^{3} \begin{bmatrix} \cos(-nx\frac{\pi}{2}) & -\sin(-nx\frac{\pi}{2}) \\ \sin(-nx\frac{\pi}{2}) & \cos(-nx\frac{\pi}{2}) \end{bmatrix} \begin{bmatrix} i_f(n,y) \\ q_f(n,y) \end{bmatrix} \qquad (6\text{-}4)$$

x=1,2,3,4　y=1,2,3,4

i'とq'はそれぞれ2次元FFTの中間結果を表わし、さらに式(6-5)によってY軸に沿ったFFTが計算され、ビームのデータが得られる。

$$\begin{bmatrix} I(X,Y) \\ Q(X,Y) \end{bmatrix} = \sum_{n=0}^{3} \begin{bmatrix} \cos(-ny\frac{\pi}{2}) & -\sin(-ny\frac{\pi}{2}) \\ \sin(-ny\frac{\pi}{2}) & \cos(-ny\frac{\pi}{2}) \end{bmatrix} \begin{bmatrix} i'(x,n) \\ q'(x,n) \end{bmatrix} \qquad (6\text{-}5)$$

x=1,2,3,4　y=1,2,3,4　X=1,2,3,4　Y=1,2,3,4

$I(X,Y)$および$Q(X,Y)$はそれぞれ、合成されたビームであり、X、Yはビームの座標(表6-1.)を表わす。試作したDBFアンテナシステムのアンテナ素子は搬送波周波数の$\lambda/2$の間隔で配置しているので、16本のビームはそれぞれ図6-2.に示される方向からのビームに相当する。また本DBFアンテナシステムは振幅比較器を持ち、これら16本のビームのうち、最も振幅の大きいビームを自動的に選択する事ができる。すなわちジャイロ等の方向センサなしに最大受信ビームを自動的に追尾出来る機能を有する事を示している。

FFTアルゴリズムによってマルチビームを形成するとそれぞれのビームの位相中心が異なり、ビームの切り替え時において位相不連続が生じる。これを避けるため、選択されたビームに対し式(6-6)に示される移相量の位相回転(式(6-7))を与える。

$$\phi_{xy} = \exp(-j(\frac{3\pi(x-1)}{4}))\exp(-j(\frac{3\pi(y-1)}{4})) \qquad (6\text{-}6)$$

$$\begin{bmatrix} I'_{xy} \\ Q'_{xy} \end{bmatrix} = \begin{bmatrix} \cos(\phi_{xy}) & -\sin(\phi_{xy}) \\ \sin(\phi_{xy}) & \cos(\phi_{xy}) \end{bmatrix} \begin{bmatrix} I_{xy} \\ Q_{xy} \end{bmatrix} \qquad (6\text{-}7)$$

x=0,1,2,3　y=0,1,2,3

ただし、表2の座標で、X=4 は、x=0、Y=4 は、y=0である。

表6-1 FFT演算のxy軸と各ビームの関係

Y\X	1	2	3	4
1	beam0	beam4	beam8	beam12
2	beam1	beam5	beam9	beam13
3	beam2	beam6	beam10	beam14
4	beam3	beam7	beam11	beam15

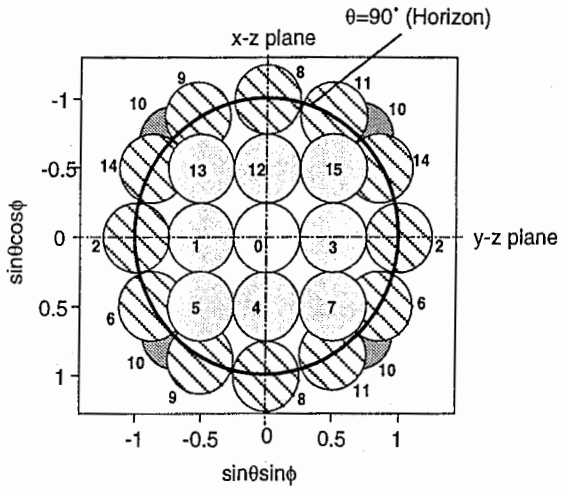


図6-2. ビームの方向とビーム番号

第2節 Multibeam部

マルチビーム生成部はDSPボード上のu1～u8に構築されている。具体的な回路は付録の回路図u1を参照して頂きたい。u1とその他のFPGAの回路上の違いはローカルコントローラとしてのスレーブシーケンサ(u1、sheet1)、FFTの係数用ROMの内容、ビームセレクタによって選択されたイネーブル信号をデコードする部分(u1、sheet15)である。イネーブル信号をデコードする部分についてはu1 sheet15の記述された回路中で、8bitのデコーダの出力をFPGAの番号により接続を違えているだけである。

第1項 Components

1. 8bit 固定小数点プロセッサ

Multibeam部での演算はすべて8-bitの固定小数点精度で行われている。実現できるハードウェア規模に余裕がある場合、スケーリングなどに注意すれば、精度を上げる事も当然できる。8-bit精度は使用したFPGAのゲート数の制限により決まった。

1-1 マルチプライヤ部(Multiplier部)

Multiplier部は2つの8-bitデータを入力とする1の演算を行なう部分と16bitのfull adderから構成されている。DSP内では、特に断らない限り2の補数形式のデータを扱うものとする。

1. A × B = Carry値と演算値
2. Carry値＋演算値

このマルチプライヤ部の1は完全な組み合わせ回路で実現されている。FPGAの開発ツールにはこの部分のlibraryがないので、VHDLで記述しそれを論理合成ツールを用いて回路に変換し用いている。また、特に本FPGAの設計のプロセッサ部の動作速度はこの部分で決まっていると言っても良いと考えられる。これは、Accumulatorのように完全に最適化されたライブラリとして配置配線できないためである。2の部分はLibraryとして供給されている16bit Adderを用いた。

Scaling

DSPの固定小数点演算にはスケーリングは付き物であり、通常、積算の後は結果の上位ビットを用いる。この時に、データの精度を確保するため最適なスケーリングを行なう。例えば、8bit×8bitの演算の結果は16bitデータが得られる、この時に上位の8bit目から15bit目までを結果とするか、7bit目から14bit目までを結果とするかを決定する事である。この部分をアダプティブに制御する方法もあるが簡単のため試作DSPでは固定のスケーリングファクタにした。

Overflow clip

入力信号が大い場合などに、ダイナミックレンジを越えるような計算結果が得られたり、その精度で表現できる数よりも大きくなってしまった場合、オーバーフローになる。オーバーフローになれば符号が反転しその数値も反転してしまう。これを避けるための方法として上位ビットと符号ビットをモニタしそれを越えた場合、最上値あるいは最下値にすればよいが、試作DSPでは簡単なゲートを組み合わせてオーバーフローのモニタを行なっている。

Rounding error

丸め誤差はスケーリングを行なう時に生じる。スケーリング時にただ上位ビットを次段に送った場合、2の補数形式ではマイナスにオフセットがかかってしまう。これを避けるために切り捨てるビットの最上位で0捨1入の処理を行なう。これを実現するテクニックとしては、
1. 切り捨て前のコンポーネントの出力に必要なビット数+1のアダーを配置し、最下位ビットに1を足す。
2. AccumulatorのLoad機能を利用する。

1-2　16bit Accumulator部

積和演算の和の部分を行なうComponentである。Xilinxのlibraryの16bit Accumulatorをそのまま使用している。各演算グループの開始直前には必ずResetあるいは丸め誤差改善用のデータをAccumulatorにロードしておき使用する。ResetあるいはLoad信号はsequencerから制御される。

1-3　Pipelining

プロセッサ部は1 MCLKですべての処理がなされるのではなく、中間レジスタを用いて3段のパイプライン構成で処理される。パイプライン処理のタイミングを図6-3に示す。試作DSPでのシーケンスについて説明する。システム動作速度を上げる必要があれば、さらに中間レジスタを増やす方法もある。しかし、その分必要となるハードウェア規模も大きくなる。

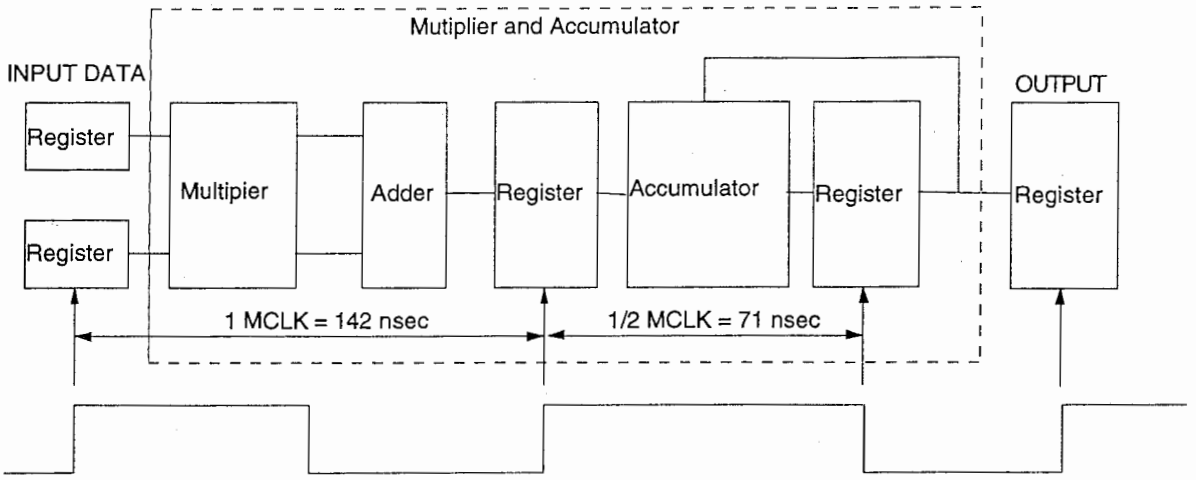


図6-3. プロセッサ部のパイプライン処理

1. MCLKの立上がりエッジでプロセッサ直前の入力レジスタに2つのデータがラッチされる。
2. 次のMCLKの立上がりまでの間に積算を完了させる。
3. 積算結果はMCLKの立上がりエッジで、中間レジスタにラッチされる。
4. その結果はAccumulatorの入力に渡され、Accumulator内のレジスタの数値と和減算が行なわれる。
5. 4の積和結果はMCLKの立ち下がりエッジでAccumulator内のレジスタにラッチされる。

以上のシーケンスで、1回の積和演算が1.5MCLKで完了する。パイプライン処理は演算の速度を向上させる事ができるが、中間レジスタによって回路規模の増大になり、また直前の演算結果を用いて次の演算を行なう処理などは待ち時間が生じてしまう欠点がある。逆に言えば、FIRフィルタのように連続して積和演算を行なうようなアルゴリズムは効率がよい。最終的な結果は5の演算完了後のMCLKの立上がりエッジで出力用のレジスタにラッチする。

1-4 Output & intermidiate register

これは、Accumulatorからの出力を1時的に保持するためのレジスタの事を指す。通常MCLKの立上がりエッジでラッチされる。ラッチの際のトリガ信号は各sequencerから送り出される。トリガ信号の遅延量の影響を避けるため、レジスタのトリガイネーブル端子にトリガ信号は与えられ、トリガイネーブル状態でレジスタのクロック入力端子に繋がれたClock信号のエッジでラッチは行われる。

1-5 Multiplexer

入力切り替えのためのコンポーネントであり、制御信号は各sequencerで発生される。またMultiplexerの替わりにトライステートのバッファとバスを用いて入力切り替えはできる。試作DSPでは使用できるBUSの本数に制限があったので、BUSの必要な回路部分(CPU I/Fのデータバスなど)を優先して設計したため、プロセッサへの入力はMultiplexerを用いている。

1-6 Input register

プロセッサ直前のレジスタや、外部からのデータを保持するレジスタの事を指す。

1-7 RAM

FIR RAM

FIRフィルタを構築する際に必要となるFIFO(Fast In Fast Out)の構成を持つRAMである。実際はFIFOでは、RAMは普通の構成のものを使用し、sequencerのaddress発生回路で制御して実現している。試作DSPはtap数10段FIRフィルタなので、RAMのaddressの深さは9である。

LOCAL RAM

各Multibeam生成用FPGA(u1～u8)で使用されている。準同期検波用のディジタルオシレータとしてのデータである。通常は128kHzサンプルでIF周波数32kHzの場合、復素平面の軸上を移動する単位ベクトルである。しかし試作DSPではこのベクトルを利用してアナログ部で生じる各素子からの信号の振幅、位相を補正しているので、CPU I/F経由で随時書き換え可能なようにRAMを用いている。

1-8　ROM

FIR ROM

FIRフィルタの係数を保持しているROMである。u10内にあり、FIRフィルタリング時に係数データをu1～u8へ分配する。

FFT ROM

FFT演算の係数を保持しているROMである。u1～u8内にありチャンネルにより値が異なる。

1-9　Controller

すべてのプロセッサ、レジスタ、マルチプレクサ、RAM、ROMのコントロールを司るコンポーネントである。これらはすべてVHDLで記述され、MCLKに同期して動作している。またstart_clkがこれらのsequencerの動作開始基準信号として扱われる。

Master sequencer

u10にあり、主にu1～u8の制御で共通している制御信号を発生するのがMaster sequencer(MSEQ)である。Master sequencerが取り扱う制御信号は、以下の通りである。

1. 準同期検波演算、FIR演算、FFT演算、ビーム電力での演算結果を保持するレジスタのトリガ信号(OUT_TRIG)
2. Accumulatorのリセット信号 (OUT_EN(2))
3. 電力値演算用マルチプレクサ切り替え信号(OUT_EN(1))
4. FIR用のFIFO RAM用のライトイネーブル信号 (OUT_EN(0)
5. FIR、FFT、電力計算の計算タイミングを表わすステート信号 (FIR_STATE, FFT1_STATE, FFT2_STATE, SPEC_STATE)
6. FIR係数ROMのアドレス発生 (FIR_ADD)

Slave sequencer

各u1～u8で主にそれぞれローカルに異なる制御を取り扱うものをslave sequencer(SSEQ)と呼ぶ事にする。各SSEQで取り扱う制御信号は、

1. 各種マルチプレクサの切り替え信号 (FFT_SEL, INPUT_DATA_SEL, FIR_SEL, IQ_RAM_SEL)
2. FIR用のFIFO RAM用のアドレスデータ発生 (RD_RAM_ADD)
3. 準同期検波用ローカル信号用RAMのアドレスデータ発生 (LOCAL_WN_ADD)

4. FFT用の係数ROMのアドレスデータ発生 (FFT_WN_ADD)
5. 入力A/Dデータモニタ用コントロール (RD_CONT)
6. FFT BUS入出力切り替え用信号(OUT_EN)

第2項　Calibration

アナログ部で発生する素子毎の位相差、振幅差はディジタル部で補正される。実際の補正は準同期検波時に入力信号に掛けられる係数に事前に測定された位相、振幅データから計算された値を用いて行われる。

第3項　FFT Multibeam Formingアルゴリズム概要　(演算シーケンス)

実際のFPGA内で行われている演算について述べる。まず、入力から準同期検波、FIRフィルタリングまでの演算について表6-2に示す。これらの演算は各素子で共通であり、4回を1周期として同じ演算が繰り返されている。この表中にはでは4回の繰り返し分が書かれている。よってそれぞれの演算後にはベースバンドなったIch, Qch成分が得られる。表中a0～a9はFIRフィルタの係数であり、Local(0)～Local(7)はCalibrationが考慮された、準同期検波用ディジタルオシレータの値である。添え字が偶数の場合にはIch成分のデータ、奇数の場合にはQch成分のデータが入っている。続いて2次元FFTと電力値の演算が行われる。
表6-3から表6-6は2次元FFT演算での分散処理のデータの流れを示したものである。FFT時の特徴としては、各DSPが持っているデータを順番にBUS上に出力し、そのデータを用いて各DSPで計算を行う。出力されたデータをタイムリーにバスが繋がっているDSPが使用できるように、計算の順番を考慮してある。BUS構造を採用した利点は、必要なデータの受け渡しに、各々配線を行えばその数は大きくなるが各DSPの出力部にトライステートバッファを用いたBUS構造にすれば配線数は1/(必要なDSP間のデータ数)になる。実際にはこのDSPではFFTではなくDFTのアルゴリズムで演算が使われている。4素子の場合、FFTとDFTに必要な演算回数は同じになり、データの取り扱い方のみが違ってくる。他素子のアレーを扱う場合、この4素子のケースを拡張して行けば、FFTアルゴリズムに対応する。つまり4素子を1クラスタとして扱いバタフライ演算を行なえばよい。
以下に詳細演算シーケンスについての説明を行う。
1.1st FFTは2軸のうち1つの方向について行い、2nd FFTはもう1方について演算するフェイズである。channelは図6-2に示したものと同等であり、それぞれのchannelはI成分とQ成分を持つ。
2.Raw Addは右隣のRaw DataがストアーされているDSPのchannelが示されている。つまりI(0)の所在はU1のch0-0というレジスタである事を示している。
3.X Data Addは、各channelでの演算に必要なデータの場所を示している。
4.FFT X DataはX Data Addで示された場所にあるデータで、4つのデータごとに実数部と虚数部が演算される。
5.Wxは左隣のFFT X Dataと掛け合わせる回転子である。
6.X Addは1st FFTの結果が収納されるレジスタのアドレスを示している。
7.X-syn dataは1st FFTの結果のデータ名を示す。
8. channel (2nd FFT)は図2で直交するBUSを用いて演算するchannelなので1st FFTと異なっている。
9. Y Data Addは2nd FFTの演算で必要なデータのアドレスを示す。
10. FFT Y DataはY Data Addで示されたアドレスにあるデータである。
11. Wy は左隣のFFT Y Dataと掛け合わせる回転子である。
12. XY Addは2nd FFTの結果が収納されるレジスタのアドレスを示している。
13. X,Y-syn Dataは2次元FFTによって合成されたデータである。
1st FFT、2nd FFTともにそれぞれ必要なデータを持つ4つのchannelがBUSを持ち、データの受け渡しを行う。
本方式の場合、16チャンネル分の2次元FFTの演算は16回の積和演算で行える。

表6-2 演算のフロー(準同期検波からFIRフィルタリングまで)

	Channel	入力データ	1st period 係数データ	2nd period 係数データ	3rd period 係数データ	4th period 係数データ
準同期検波乗算	I	A/D Data(0)	x Local(0)	x Local(2)	x Local(4)	x Local(6)
フィルタリング	I	I Data(1)	x a0	x a0	x a0	x a0
フィルタリング	I	I Data(2)	x a1	x a1	x a1	x a1
フィルタリング	I	I Data(3)	x a2	x a2	x a2	x a2
フィルタリング	I	I Data(4)	x a3	x a3	x a3	x a3
フィルタリング	I	I Data(5)	x a4	x a4	x a4	x a4
フィルタリング	I	I Data(6)	x a5	x a5	x a5	x a5
フィルタリング	I	I Data(7)	x a6	x a6	x a6	x a6
フィルタリング	I	I Data(8)	x a7	x a7	x a7	x a7
フィルタリング	I	I Data(9)	x a8	x a8	x a8	x a8
フィルタリング	I	I Data(0)	x a9	x a9	x a9	x a9
準同期検波乗算	Q	A/D Data(0)	x Local(1)	x Local(3)	x Local(5)	x Local(7)
フィルタリング	Q	Q Data(1)	x a0	x a0	x a0	x a0
フィルタリング	Q	Q Data(2)	x a1	x a1	x a1	x a1
フィルタリング	Q	Q Data(3)	x a2	x a2	x a2	x a2
フィルタリング	Q	Q Data(4)	x a3	x a3	x a3	x a3
フィルタリング	Q	Q Data(5)	x a4	x a4	x a4	x a4
フィルタリング	Q	Q Data(6)	x a5	x a5	x a5	x a5
フィルタリング	Q	Q Data(7)	x a6	x a6	x a6	x a6
フィルタリング	Q	Q Data(8)	x a7	x a7	x a7	x a7
フィルタリング	Q	Q Data(9)	x a8	x a8	x a8	x a8
フィルタリング	Q	Q Data(0)	x a9	x a9	x a9	x a9

表6-3 演算のフロー(FFTから電力演算まで)

1st FFT								2nd FFT					
channel	Raw Add	Raw Data	X Data Add	FFT X Data	Wx	X Add	X-syn data	channel	Y Data Add	FFT Y Data	Wy	XY Add	X,Y-syn Data
ch0	ch0-0	I(0)	ch0-0	I(0)	x1	ch0-2	I'(0)	ch0	ch0-2	I'(0)	x1	ch0-4	I''(0)
			ch1-0	I(1)	x1				ch4-2	I'(4)	x1		
			ch2-0	I(2)	x1				ch8-2	I'(8)	x1		
			ch3-0	I(3)	x1				ch12-2	I'(12)	x1		
	ch0-1	Q(0)	ch0-1	Q(0)	x1	ch0-3	Q'(0)		ch0-3	Q'(0)	x1	ch0-5	Q''(0)
			ch1-1	Q(1)	x1				ch4-3	Q'(4)	x1		
			ch2-1	Q(2)	x1				ch8-3	Q'(8)	x1		
			ch3-1	Q(3)	x1				ch12-3	Q'(12)	x1		
ch1	ch1-0	I(1)	ch0-1	Q(0)	x1	ch1-2	I'(1)	ch4	ch0-3	Q'(0)	x1	ch4-4	I''(4)
			ch1-0	I(1)	x(-1)				ch4-2	I'(4)	x(-1)		
			ch2-1	Q(2)	x(-1)				ch8-3	Q'(8)	x(-1)		
			ch3-0	I(3)	x1				ch12-2	I'(12)	x1		
	ch1-1	Q(1)	ch0-0	I(0)	x(-1)	ch1-3	Q'(1)		ch0-2	I'(0)	x(-1)	ch4-5	Q''(4)
			ch1-1	Q(1)	x(-1)				ch4-3	Q'(4)	x(-1)		
			ch2-0	I(2)	x1				ch8-2	I'(8)	x1		
			ch3-1	Q(3)	x1				ch12-3	Q'(12)	x1		
ch2	ch2-0	I(2)	ch0-0	I(0)	x(-1)	ch2-2	I'(2)	ch8	ch0-2	I'(0)	x(-1)	ch8-4	I''(8)
			ch1-0	I(1)	x1				ch4-2	I'(4)	x1		
			ch2-0	I(2)	x(-1)				ch8-2	I'(8)	x(-1)		
			ch3-0	I(3)	x1				ch12-2	I'(12)	x1		
	ch2-1	Q(2)	ch0-1	Q(0)	x(-1)	ch2-3	Q'(2)		ch0-3	Q'(0)	x(-1)	ch8-5	Q''(8)
			ch1-1	Q(1)	x1				ch4-3	Q'(4)	x1		
			ch2-1	Q(2)	x(-1)				ch8-3	Q'(8)	x(-1)		
			ch3-1	Q(3)	x1				ch12-3	Q'(12)	x1		
ch3	ch3-0	I(3)	ch0-1	Q(0)	x(-1)	ch3-2	I'(3)	ch12	ch0-3	Q'(0)	x(-1)	ch12-4	I''(12)
			ch1-0	I(1)	x(-1)				ch4-2	I'(4)	x(-1)		
			ch2-1	Q(2)	x1				ch8-3	Q'(8)	x1		
			ch3-0	I(3)	x1				ch12-2	I'(12)	x1		
	ch3-1	Q(3)	ch0-0	I(0)	x1	ch3-3	Q'(3)		ch0-2	I'(0)	x1	ch12-5	Q''(12)
			ch1-1	Q(1)	x(-1)				ch4-3	Q'(4)	x(-1)		
			ch2-0	I(2)	x(-1)				ch8-2	I'(8)	x(-1)		
			ch3-1	Q(3)	x1				ch12-3	Q'(12)	x1		

表6-4. 各チャンネルにおけるFFT演算のフロー(その2)

ch8	ch8-0	I(8)	ch8-0	I(8)	x1	ch8-2	I'(8)	ch1	ch1-2	I'(1)	x1	ch1-4	I''(1)
			ch9-0	I(9)	x1				ch5-2	I'(5)	x1		
			ch10-0	I(10)	x1				ch9-2	I'(9)	x1		
			ch11-0	I(11)	x1				ch13-2	I'(13)	x1		
	ch8-1	Q(8)	ch8-1	Q(8)	x1	ch8-3	Q'(8)		ch1-3	Q'(1)	x1	ch1-5	Q''(1)
			ch9-1	Q(9)	x1				ch5-3	Q'(5)	x1		
			ch10-1	Q(10)	x1				ch9-3	Q'(9)	x1		
			ch11-1	Q(11)	x1				ch13-3	Q'(13)	x1		
ch9	ch9-0	I(9)	ch8-1	Q(8)	x1	ch9-2	I'(9)	ch5	ch1-3	Q'(1)	x1	ch5-4	I''(5)
			ch9-0	I(9)	x(-1)				ch5-2	I'(5)	x(-1)		
			ch10-1	Q(10)	x(-1)				ch9-3	Q'(9)	x(-1)		
			ch11-0	I(11)	x1				ch13-2	I'(13)	x1		
	ch9-1	Q(9)	ch8-0	I(8)	x(-1)	ch9-3	Q'(9)		ch1-2	I'(1)	x(-1)	ch5-5	Q''(5)
			ch9-1	Q(9)	x(-1)				ch5-3	Q'(5)	x(-1)		
			ch10-0	I(10)	x1				ch9-2	I'(9)	x1		
			ch11-1	Q(11)	x1				ch13-3	Q'(13)	x1		
ch10	ch10-0	I(10)	ch8-0	I(8)	x(-1)	ch10-2	I'(10)	ch9	ch1-2	I'(1)	x(-1)	ch9-4	I''(9)
			ch9-0	I(9)	x1				ch5-2	I'(5)	x1		
			ch10-0	I(10)	x(-1)				ch9-2	I'(9)	x(-1)		
			ch11-0	I(11)	x1				ch13-2	I'(13)	x1		
	ch10-1	Q(10)	ch8-1	Q(8)	x(-1)	ch10-3	Q'(10)		ch1-3	Q'(1)	x(-1)	ch9-5	Q''(9)
			ch9-1	Q(9)	x1				ch5-3	Q'(5)	x1		
			ch10-1	Q(10)	x(-1)				ch9-3	Q'(9)	x(-1)		
			ch11-1	Q(11)	x1				ch13-3	Q'(13)	x1		
ch11	ch11-0	I(11)	ch8-1	Q(8)	x(-1)	ch11-2	I'(11)	ch13	ch1-3	Q'(1)	x(-1)	ch13-4	I''(13)
			ch9-0	I(9)	x(-1)				ch5-2	I'(5)	x(-1)		
			ch10-1	Q(10)	x1				ch9-3	Q'(9)	x1		
			ch11-0	I(11)	x1				ch13-2	I'(13)	x1		
	ch11-1	Q(11)	ch8-0	I(8)	x1	ch11-3	Q'(11)		ch1-2	I'(1)	x1	ch13-5	Q''(13)
			ch9-1	Q(9)	x(-1)				ch5-3	Q'(5)	x(-1)		
			ch10-0	I(10)	x(-1)				ch9-2	I'(9)	x(-1)		
			ch11-1	Q(11)	x1				ch13-3	Q'(13)	x1		

表6-5. 各チャンネルにおけるFFT演算のフロー(その3)

ch4	ch4-0	I(4)	ch4-0	I(4)	x 1	ch4-2	I'(4)	ch2	ch2-2	I'(2)	x 1	ch2-4	I''(2)
			ch5-0	I(5)	x 1				ch6-2	I'(6)	x 1		
			ch6-0	I(6)	x 1				ch10-2	I'(10)	x 1		
			ch7-0	I(7)	x 1				ch14-2	I'(14)	x 1		
	ch4-1	Q(4)	ch4-1	Q(4)	x 1	ch4-3	Q'(4)		ch2-3	Q'(2)	x 1	ch2-5	Q''(2)
			ch5-1	Q(5)	x 1				ch6-3	Q'(6)	x 1		
			ch6-1	Q(6)	x 1				ch10-3	Q'(10)	x 1		
			ch7-1	Q(7)	x 1				ch14-3	Q'(14)	x 1		
ch5	ch5-0	I(5)	ch4-1	Q(4)	x 1	ch5-2	I'(5)	ch6	ch2-3	Q'(2)	x 1	ch6-4	I''(6)
			ch5-0	I(5)	x(-1)				ch6-2	I'(6)	x(-1)		
			ch6-1	Q(6)	x(-1)				ch10-3	Q'(10)	x(-1)		
			ch7-0	I(7)	x 1				ch14-2	I'(14)	x 1		
	ch5-1	Q(5)	ch4-0	I(4)	x(-1)	ch5-3	Q'(5)		ch2-2	I'(2)	x(-1)	ch6-5	Q''(6)
			ch5-1	Q(5)	x(-1)				ch6-3	Q'(6)	x(-1)		
			ch6-0	I(6)	x 1				ch10-2	I'(10)	x 1		
			ch7-1	Q(7)	x 1				ch14-3	Q'(14)	x 1		
ch6	ch6-0	I(6)	ch4-0	I(4)	x(-1)	ch6-2	I'(6)	ch10	ch2-2	I'(2)	x(-1)	ch10-4	I''(10)
			ch5-0	I(5)	x 1				ch6-2	I'(6)	x 1		
			ch6-0	I(6)	x(-1)				ch10-2	I'(10)	x(-1)		
			ch7-0	I(7)	x 1				ch14-2	I'(14)	x 1		
	ch6-1	Q(6)	ch4-1	Q(4)	x(-1)	ch6-3	Q'(6)		ch2-3	Q'(2)	x(-1)	ch10-5	Q''(10)
			ch5-1	Q(5)	x 1				ch6-3	Q'(6)	x 1		
			ch6-1	Q(6)	x(-1)				ch10-3	Q'(10)	x(-1)		
			ch7-1	Q(7)	x 1				ch14-3	Q'(14)	x 1		
ch7	ch7-0	I(7)	ch4-1	Q(4)	x(-1)	ch7-2	I'(7)	ch14	ch2-3	Q'(2)	x(-1)	ch14-4	I''(14)
			ch5-0	I(5)	x(-1)				ch6-2	I'(6)	x(-1)		
			ch6-1	Q(6)	x 1				ch10-3	Q'(10)	x 1		
			ch7-0	I(7)	x 1				ch14-2	I'(14)	x 1		
	ch7-1	Q(7)	ch4-0	I(4)	x 1	ch7-3	Q'(7)		ch2-2	I'(2)	x 1	ch14-5	Q''(14)
			ch5-1	Q(5)	x(-1)				ch6-3	Q'(6)	x(-1)		
			ch6-0	I(6)	x(-1)				ch10-2	I'(10)	x(-1)		
			ch7-1	Q(7)	x 1				ch14-3	Q'(14)	x 1		

表6-6. 各チャンネルにおけるFFT演算のフロー(その4)

ch12	ch12-0	I(12)	ch12-0	I(12)	x1	ch12-2	I'(12)	ch3	ch3-2	I'(3)	x1	ch3-4	I''(3)
			ch13-0	I(13)	x1				ch7-2	I'(7)	x1		
			ch14-0	I(14)	x1				ch11-2	I'(11)	x1		
			ch15-0	I(15)	x1				ch15-2	I'(15)	x1		
	ch12-1	Q(12)	ch12-1	Q(12)	x1	ch12-3	Q'(12)		ch3-3	Q'(3)	x1	ch3-5	Q''(3)
			ch13-1	Q(13)	x1				ch7-3	Q'(7)	x1		
			ch14-1	Q(14)	x1				ch11-3	Q'(11)	x1		
			ch15-1	Q(15)	x1				ch15-3	Q'(15)	x1		
ch13	ch13-0	I(13)	ch12-1	Q(12)	x1	ch13-2	I'(13)	ch7	ch3-3	Q(3)	x1	ch7-4	I''(7)
			ch13-0	I(13)	x(-1)				ch7-2	I'(7)	x(-1)		
			ch14-1	Q(14)	x(-1)				ch11-3	Q'(11)	x(-1)		
			ch15-0	I(15)	x1				ch15-2	I'(15)	x1		
	ch13-1	Q(13)	ch12-0	I(12)	x(-1)	ch13-3	Q'(13)		ch3-2	I'(3)	x(-1)	ch7-5	Q''(7)
			ch13-1	Q(13)	x(-1)				ch7-3	Q'(7)	x(-1)		
			ch14-0	I(14)	x1				ch11-2	I'(11)	x1		
			ch15-1	Q(15)	x1				ch15-3	Q'(15)	x1		
ch14	ch14-0	I(14)	ch12-0	I(12)	x(-1)	ch14-2	I'(14)	ch11	ch3-2	I'(3)	x(-1)	ch11-4	I''(11)
			ch13-0	I(13)	x1				ch7-2	I'(7)	x1		
			ch14-0	I(14)	x(-1)				ch11-2	I'(11)	x(-1)		
			ch15-0	I(15)	x1				ch15-2	I'(15)	x1		
	ch14-1	Q(14)	ch12-1	Q(12)	x(-1)	ch14-3	Q'(14)		ch3-3	Q'(3)	x(-1)	ch11-5	Q''(11)
			ch13-1	Q(13)	x1				ch7-3	Q'(7)	x1		
			ch14-1	Q(14)	x(-1)				ch11-3	Q'(11)	x(-1)		
			ch15-1	Q(15)	x1				ch15-3	Q'(15)	x1		
ch15	ch15-0	I(15)	ch12-1	Q(12)	x(-1)	ch15-2	I'(15)	ch15	ch3-3	Q(3)	x(-1)	ch15-4	I''(15)
			ch13-0	I(13)	x(-1)				ch7-2	I'(7)	x(-1)		
			ch14-1	Q(14)	x1				ch11-3	Q'(11)	x1		
			ch15-0	I(15)	x1				ch15-2	I'(15)	x1		
	ch15-1	Q(15)	ch12-0	I(12)	x1	ch15-3	Q'(15)		ch3-2	I'(3)	x1	ch15-5	Q''(15)
			ch13-1	Q(13)	x(-1)				ch7-3	Q'(7)	x(-1)		
			ch14-0	I(14)	x(-1)				ch11-2	I'(11)	x(-1)		
			ch15-1	Q(15)	x1				ch15-3	Q'(15)	x1		

第4項　FFT演算のためのハードウェア構成

まずDSPボード上で持つ特徴を示す。FFT演算を行なうためには他のチップで得られた結果を交換しなければならない。このため図2-1に示したアンテナ配置で縦横並びの素子からのデータを交換する、ここで図6-4に示すようにデータバスを配している。アンテナx軸方向の並びに対しては1st FFT BUSと示したデータバスが用いられ、2次元目のy軸に沿ったFFTは2nd FFT BUSと示したデータバスが用いられる。

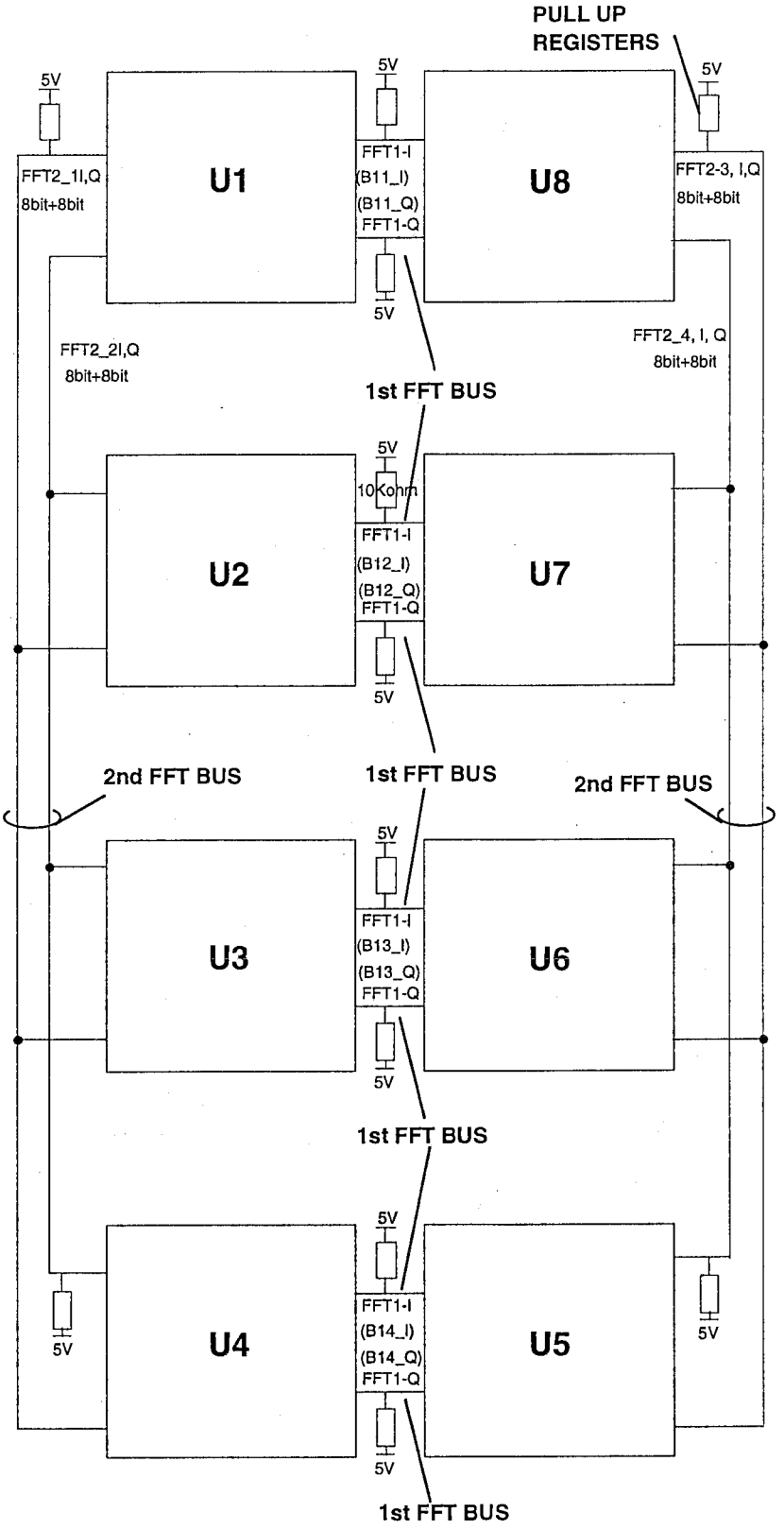


図6-4. FFT BusとFPGAの接続

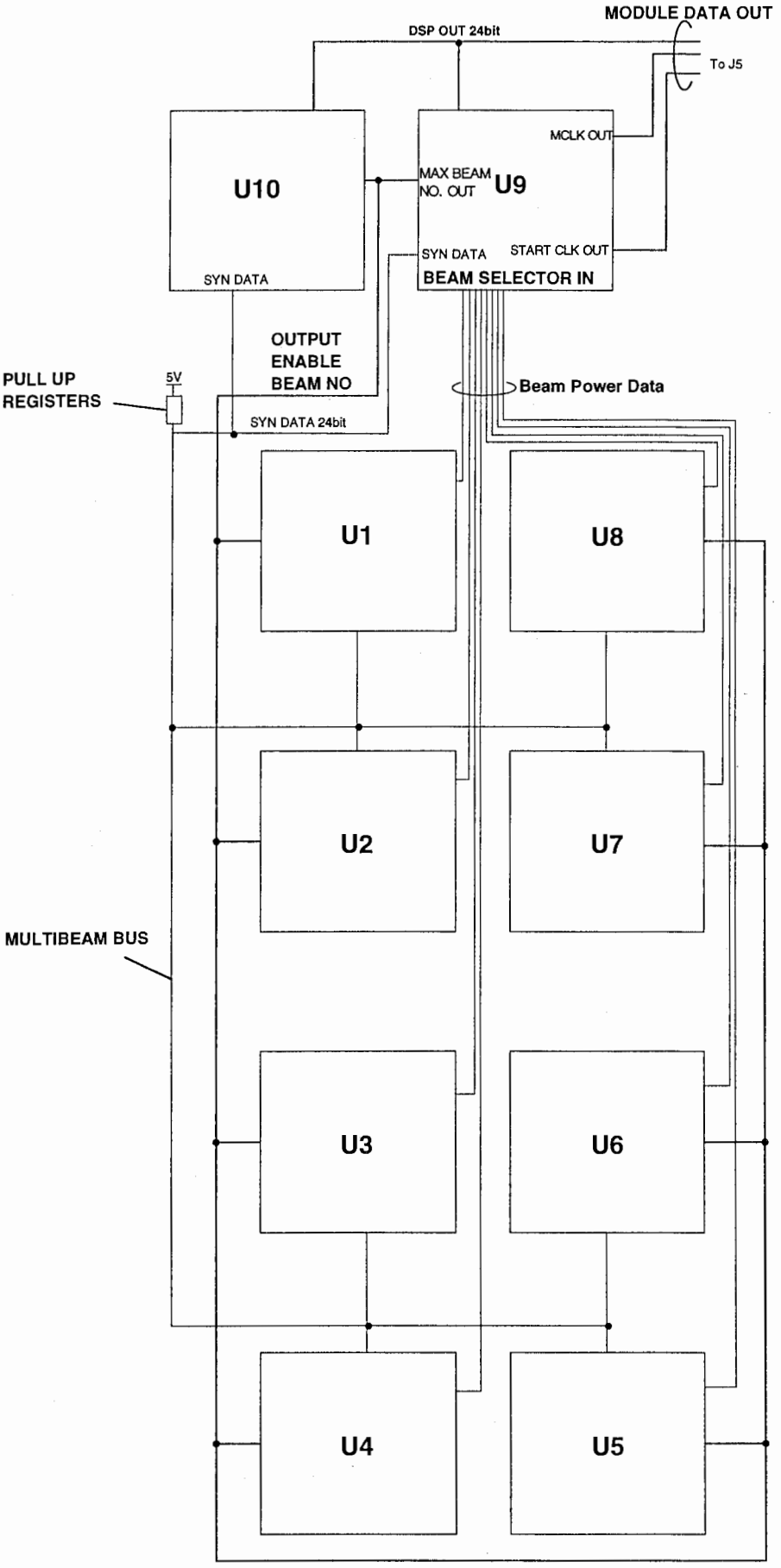


図6-5. Multibeam Bus、Beam SelectorとBeam電力のFPGA間の接続

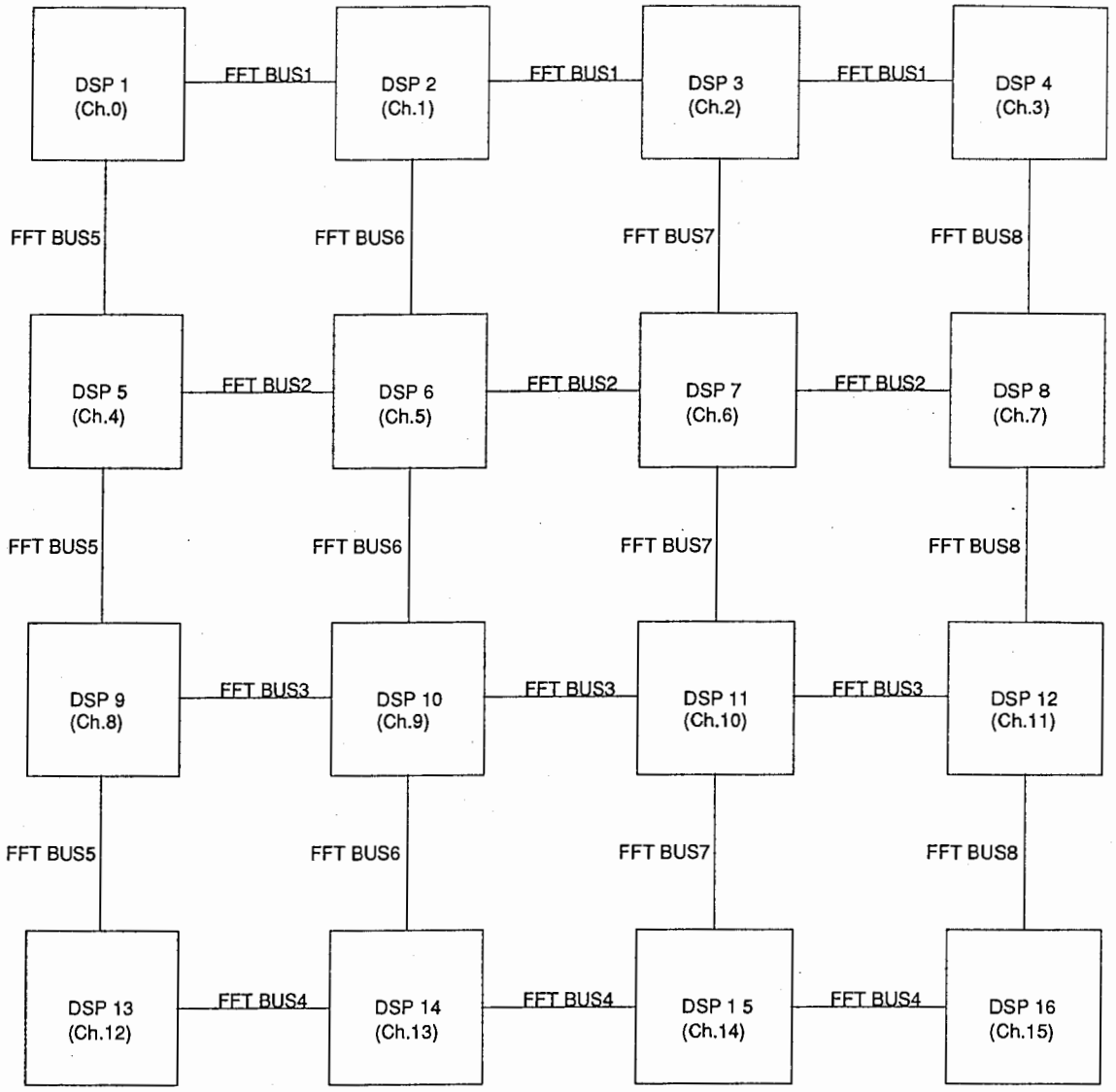


図6-6. FFT BusとProcessorの関係

図6-6は、効率的にFFT処理を行う為に考案したDSPの構成である。説明の為に4×4のアレーアンテナの配置に対応するようにDSPを並べている。それぞれX軸方向、Y軸方向にデータバスを持ち、16チャンネル分のFFTを各DSPに分散する事により同時に行える。図6-7に各素子の受信データを計算するディジタル信号処理装置のブロック図を示す。演算器の入力を適宜、制御する事によって計算され、その結果が各出力のレジスタに一時的に貯えられる。また図右側の4本のラインが2次元のフーリエ変換用のBUSである。

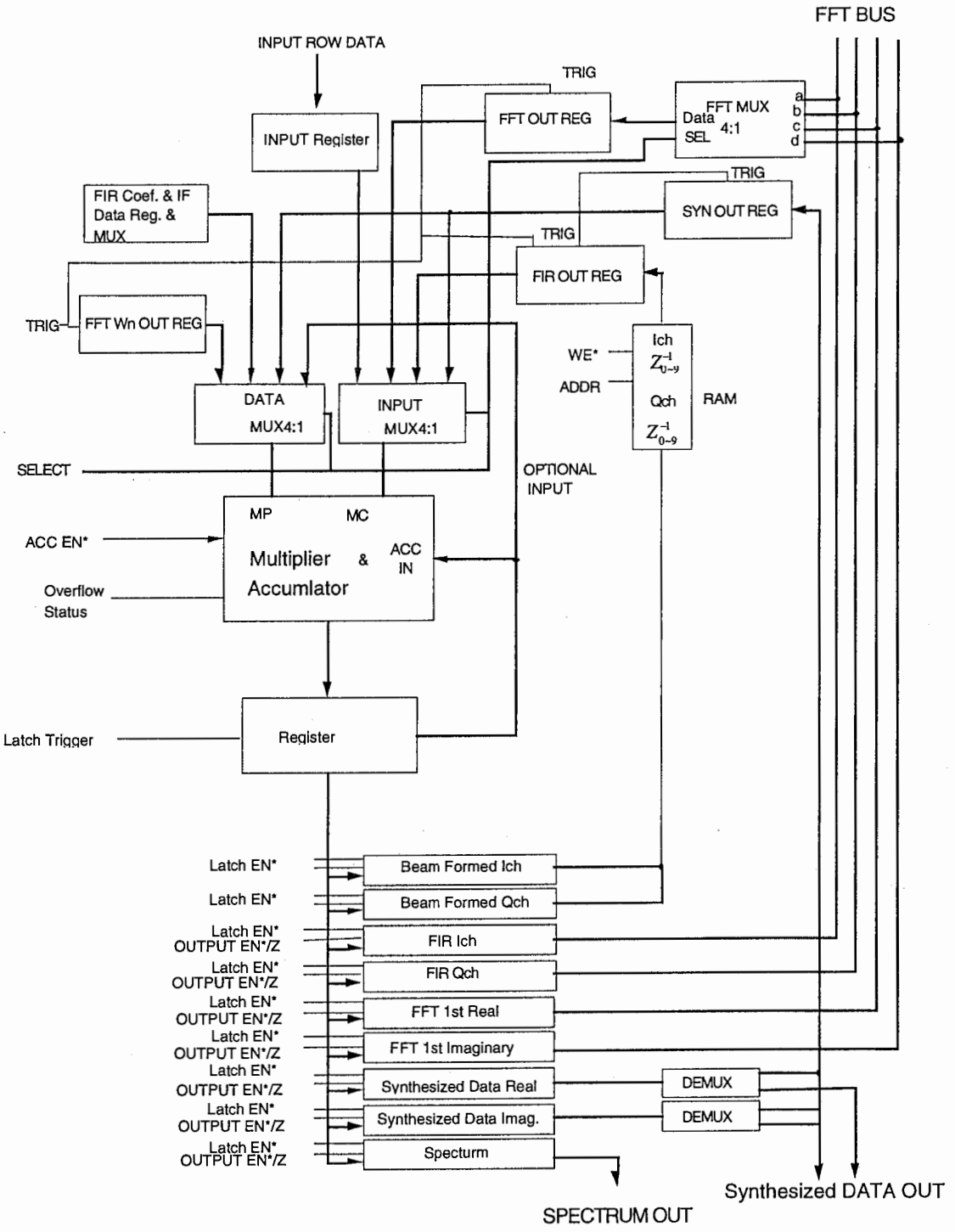


図6-7. Processor部の構成

第5項　Timing Chart

Multibeam部のTiming Chartを付録に添付する。Timing Chartでは各sequencerからの制御信号のTimingと各サンプリング周期の間に実行する演算の制御の為のフローを示している。Timing Chartに示されている制御信号について説明する。各マルチプレクサの入力データは以下の通りである。

INPUT MUX
- A: A/Dコンバータからの素子データを選択
- B: FIRフィルタリングの為過去のデータを読みだすRAMを選択
- C: FFTバスからのデータを選択
- D: 2次元FFTされた合成ビームをデータを選択

DATA MUX
- A:FIRフィルタリングの為の係数ROMを選択
- B:FFT演算の為の係数ROMを選択
- C:2次元FFTされた合成ビームを選択
- D:ローカル周波数を掛ける為の係数ROMを選択

FFT MUX
- A: FFT X軸 Ich
- B: FFT X軸 Qch
- C: FFT Y軸 Ich
- D: FFT Y軸 Qch

u10のmseqによるコントロール

OUT_EN(0): 準直交検波されたデータのFIRデータ用RAMへの書き込み信号。
OUT_EN(1): ビーム電力データを求める2乗計算のために、Ich、Qchを選択する。
OUT_EN(2): Accumulatorの内部レジスタを初期化するための信号。
OUT_TRIG(0): FIRフィルタリングされたIchデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(1): FIRフィルタリングされたQchデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(2): X軸方向のFFTがなされたIcデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(3): X軸方向のFFTがなされたQcデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(4): Y軸方向のFFTがなされたIcデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(5): Y軸方向のFFTがなされたQcデータをレジスタにストアーする為のトリガ信号。
OUT_TRIG(6): ビーム電力データをレジスタにストアーする為のトリガ信号。
OUT_TRIG(7): 準直交検波されたデータをレジスタにストアーする為のトリガ信号。
FIR係数用ROMのアドレス発生。

u1～u8のsseqによるコントロール

OUT_EN(0): FIRフィルタリングされたデータを他のチャンネルに対してバスをイネーブルにする。
OUT_EN(1): X軸FFTされたデータを他のチャンネルに対してバスをイネーブルにする。
FIR_SEL: X軸FFT時に入力データであるFIRフィルタリングされたデータのIch、Qchを選択する。
IQ_RAM_SEL: FIRフィルタリングで、当サンプル時に得られた準同期検波後の信号と過去の準同期検波されたデータを切り替える。
LOCAL用RAMのアドレス発生。
FFT用ROMのアドレス発生。

第3節　Beam selector部

第1項　Components

ここで使用したcomponentsとしては8 bit maginitude comparator、multiplexer、decoder、controllerである。Comparatorで比較する時に次段へ渡されるのはLT信号であり、入力電力値が等しい場合、チップ番号(uXX)の若い方が優先される。controllerはVHDLで記述されており、敷居値との比較のタイミングトリガの発生や2回目以降の比較のためのDisable信号、FPGA選択信号を発生する。また前回選択ビーム組み合わせと今回の選択ビーム組み合わせを比較して異なっている場合Reset用信号を発生する。

第2項　Tournament方式

Beam selector部はMagnitude comparatorを7個用いてトーナメント方式状で構成されている。複数のFPGAによりマルチビーム部は構築されている事とIO数の制限により，Beam selector部へ渡される前に，各マルチビーム用FPGA内で2つの値で比較し大きい方がBeam selectorに渡される。ビームセレクタ部での選択のシーケンスはまず1回目の比較を行い、

1. 1回目選択されたChipに対してComp_select信号を出力する。
2. Comp_select選択信号を受けたFPGAは前回出力していない方の電力値を出力する。

次に2回目の比較を行い、その選択結果によって

1. 前回に選択されたビームが過去に選択されたビームと同じチップから出力されているなら、そのFPGAに対してDisable信号を出力する。
2. もしそうでなければ、前回選択されたFPGAにComp_select信号を出力する。

の手順で選択を行なう。3回目以降同様に行なう。

IIR filter for beam amplitude

この回路は各u1～u8の電力値を保持するレジスタの後段に構築されている。

2-1　実際の電波環境でビームの電力の瞬時値を比較してのビーム選択方式では変動が大きすぎて安定な動作はできない。特に後段で処理を行う場合深刻な問題となる、そこで各ビームごとに構成が簡単な1次のIIRフィルタを用いてローパスフィルタリングを行っている。結果，非常に安定なビーム選択が実現できた。IIRフィルタのカットオフ周波数は約200Hzである。

第3項　Vth回路

設定された値より、選択されたビームの電力値が小さい場合、検知信号を発生する。検知信号はコントローラ(comp_cont)からのトリガより、フリップフロップにラッチされる。

第４項　ハードウェア構成

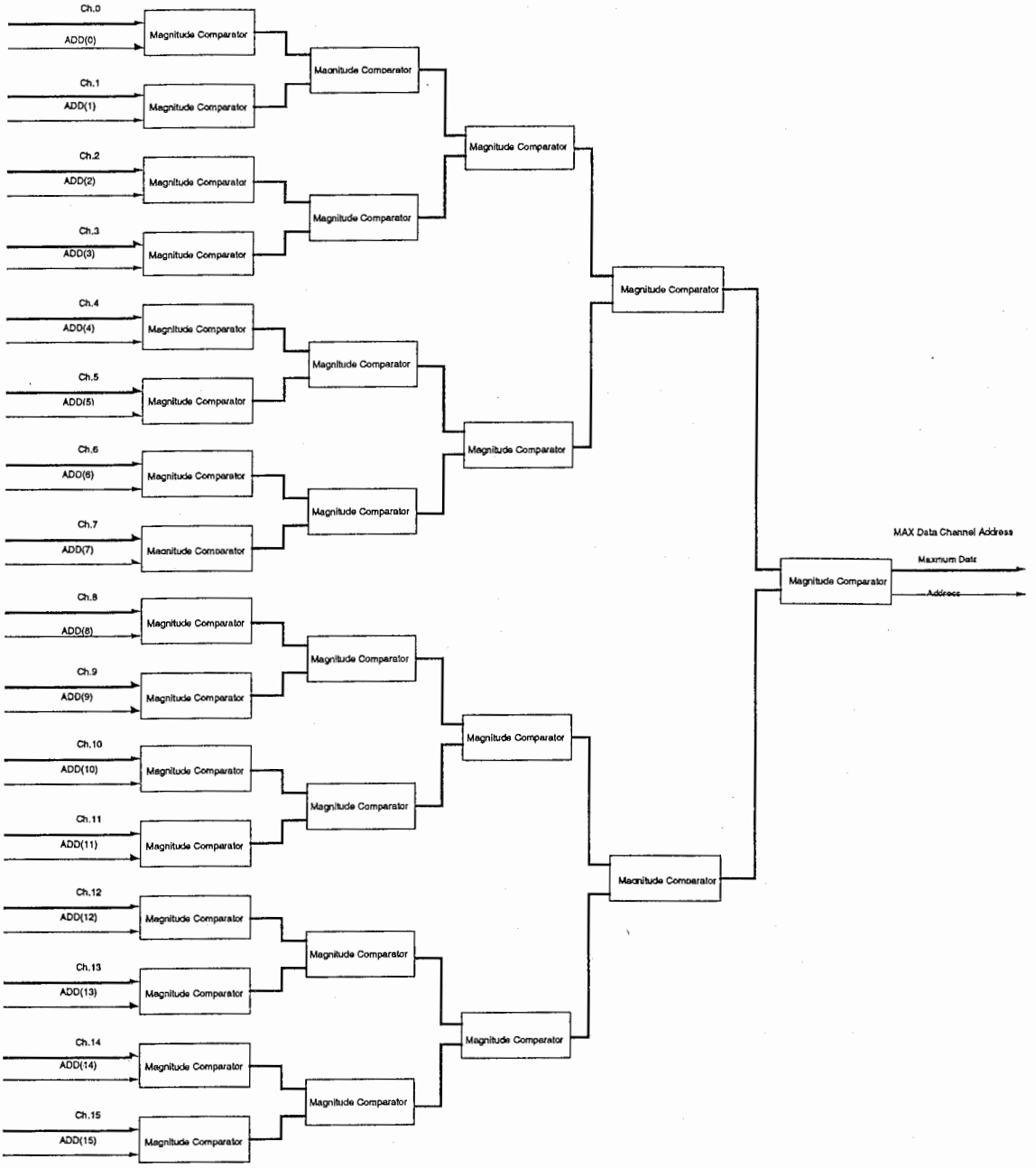


図6-8.　Beam selectorの構成

図6-8はマルチビーム選択部の構成を示している。各DSPからは、振幅データとChannelのデータが渡される。各振幅比較器で入力の最大値とチャンネルを探し出す。次に２番目３番目の大きさの振幅のデータを探す場合は、既に選ばれたチャンネルのデータは出力しないようにしておき、再度比較を行い見つけだす。

第５項　Timing Chart

Beam selector部のTiming Chartを付録に添付する。後段のCMAアダプティブ処理やSelf-Beam-Steering処理に使用するため4ビームを選択する。比較の時間は各々3MCLK分とり、その内2MCLK分が比較確定の時間である。

第4節　Vector rotation

Vector rotation部はFFT演算方式による位相中心とアンテナの物理的な位相中心の差を補正するための回路である。Vector ration部はu10に構築されている。付録回路図u10-sheet1を参照。u9で選択されたビームはビーム出力イネーブル信号としてu9からその他のFPGAに既知のタイミングでbeam番号自身が出力される。

第1項　Components

1.Processor部
このプロセッサ部はMultibeam部のプロセッサと共通の仕様である。
2. 入力データ保持用RAM
入力した最大4ビームまでのI,Qデータを保持する。データ幅8bitアドレス深さ4のRAMである。
3. beam番号保持用RAM
選択されたビーム番号を保持しておくためのRAMである。
4. 入力用Multiplexer
I,Q data切り替え、移相値Real ,Imaginary切り替え用のMultiplexer
5. 入力レジスタ
u1～u8のどれかから出力されたビームのI,Qデータを一時的に保持するためのレジスタ。ラッチ後データは入力データ保持用RAMへ書き込まれる。
6　　移相用データROM
6-1　移相量は選択されたビーム毎に決まった値であるので、ビーム番号を移相量を引き出す時に移相用データROMのアドレス値として用いている。
7　　出力レジスタ
7-1　Processorの結果を1時的に保持するためのレジスタ。ラッチ後、データは出力データ保持用RAMに書き込まれる。
8　　出力データ保持用RAM
8-1　移相された最大4ビームまでのI,Qデータを保持する。データ幅8bitアドレス深さ4のRAMである。
9　　Sequencer
9-1　上記Componentのすべての制御信号を発生するためのControllerである。VHDLで記述されている。

第2項　ハードウェア構成

Vector Rotaion部のハードウェア構成を図6-10に示す。。入力信号として、例えば大きな振幅のビームから4本選択されたとする。これらは、一時的にレジスタに保持される。同時に入力されたビームのアドレス(チャンネル番号)もレジスタに保持される。移相量としてビーム番号をアドレス値としてROMから引き出される。続いて、第6章、式(6-6)、(6-7)に示される複素演算を行い、移相された信号を得る。

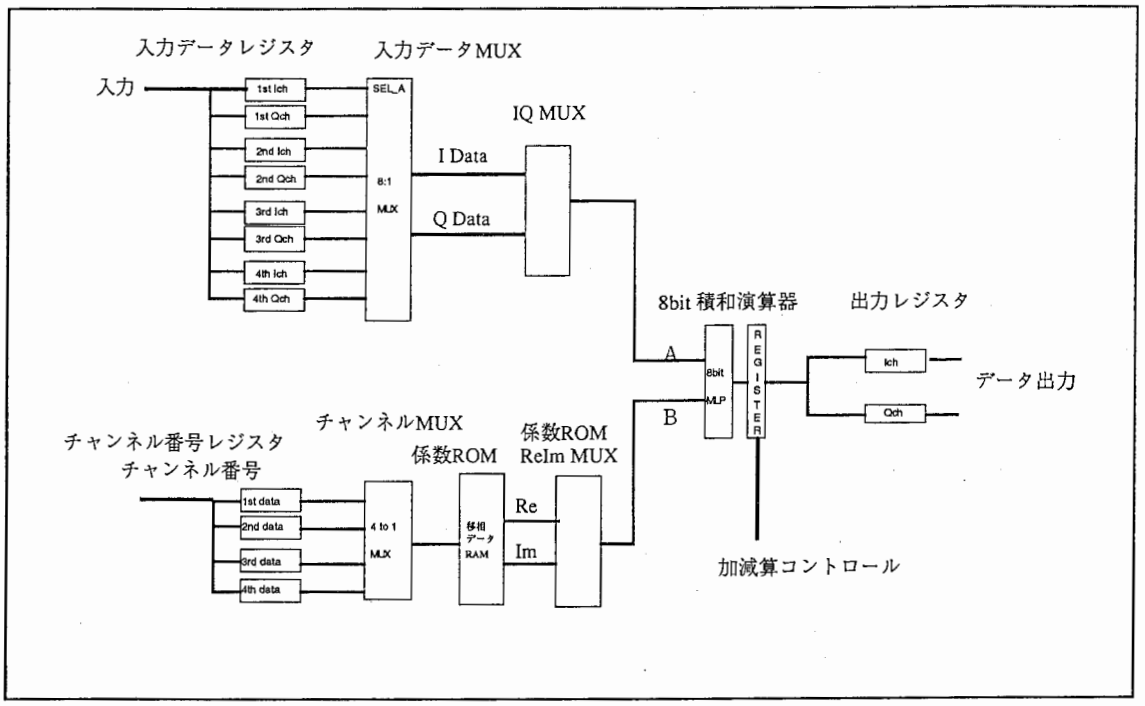


図6-9. Vector Rotation部の構成

第3項 Timing Chart

Vector Rotation部のTiming Chartを付録に添付する。選択された4ビームに対し、選択に引き続いて処理が行われる。4ベクトルの演算終了後に、4ベクトル時系列でまとめて次段へ出力される。

第5節 CPU I/F部

第1項 u10のCPU I/F部 (u10-sheet5、11、12 参照)

1 Components

1-1 Identity comparator

CPUから提示されたアドレスが自身のアドレスと一致しているかどうかを判断するために使用。

1-2 Register

自身のアドレスと一致した場合に提示アドレスを保持するためのレジスタ。

1-3 Decoder

アドレスをデコードして選択されたRead/Writeレジスタに対しイネーブル信号を出力する。

1-4 sequencer

CPUから提示されたアドレスが自身のアドレスと一致した場合、Read/WriteのEnable信号などを発生する。

Read/WriteのTiming chartに従い、VHDLで記述されている。

1-5 Read Regisiters

読み込み用のレジスタ。CPUはこのレジスタにアクセスして測定結果、ウェイト情報などを取得する。

1-6 Write Registers

書き込み用のレジスタ。CPUから電力敷居値など各種パラメータを設定するためのレジスタ。

第2項 u1～u9のCPU I/F部 (u1～u8-sheet18,19,21、u9-sheet12～14,16)

u1～u9のCPU I/F部はu10から指定信号を受けて初めて動作する。指定を受けた場合下位アドレス値(A4～A1)をデコードし、各Read/Writeレジスタに対しイネーブル信号を出力する。

1 Components

1-1 Register

u10からの指定信号を受けて、下位アドレス値(A4～A1)をラッチする。

1-2 Decoder

下位アドレス値(A4～A1)をデコードし、各Read/Writeレジスタに対しイネーブル信号を出力する。

第6節 Clock Management部 (付録回路図u10-sheet1を参照。)

第1項 Components

1 Binary counter(8bit, 4bit)

1-1 8bit counter。1 sampleイタレーションを決定するカウンタ。最大数までカウント後、TC信号をフィードバックしてプリセット値に設定される。このプリセット値が1 sampleのMCLK数を決定する。プリセット値は201に設定され55 MCLKを1sampleとする間隔で動作している。またTC信号がすべての始まりを決定するstart_clkになり分配される。

1-2 4bit counter。8bit counterのTC出力でカウントし、start_clkが4回で繰り返す。このcounterのTC出力をphase_clkとしている。

第7章 むすび

ASICを用いたDigital Beamforming Antennaの設計・試作について述べた。比較的小規模な回路構成(FGPA 10個、130,000ゲート以下)でマルチビーム生成が行なえる事が明らかとなった。試作は8bit固定小数点精度で構築したが、実用にはさらに高精度化をはかる必要があると思われる。しかしカスタムLSIを適応すればさらに大規模な構成にでき、また微細化技術による高速化も可能である。試算では、現状の技術で1Mbps以上の通信に適応できることを示した[6-1]。将来のディジタル化が通信の分野のみならず、発展して行くと思われる。よって他の分野でのディジタルデバイスの要求が牽引役ともなり、DBFアンテナの実用が現実のものとなっていく時も遠くはないと感じている。

第8章 謝辞

日頃より御指導頂く、ATR光電波通信研究所 猪股英行社長に深く感謝いたします。また本研究を進めるにあたり正しい方向へお導き頂いた、唐沢好男室長、千葉　勇元主任研究員、三浦　龍主任研究員に深く感謝いたします。また、無線通信第一研究室の皆様に感謝いたします。

第９章　参考文献

[1-1] 田中豊久, "BSCMAアダプティブアレーアンテナディジタル信号処理部の開発," ATRテクニカルレポート, TR-O-0115 (TR-O-0118、設計データソフト付き), Mar. 1996.

[1-2] 田中豊久, "DBFセルフビームステアリングアレーアンテナディジタル信号処理部の開発," ATRテクニカルレポート, TR-O-0116 (TR-O-0119、設計データソフト付き), Mar. 1996.

[2-1] H. Steyskal, "Digital Beamforming Antennas," Microwave Journal, Jan. 1987, pp107-114.

[2-2] K. Takao and K. Uchida, ﾒBeamspace Partially adaptive antenna,ﾓ IEE Proc., Pt. H, 6, pp.439-444 Dec. 1989.

[2-3] 藤元美俊、菊間信良、稲垣直樹、"マルカート法を用いたCMAアダプティブアレーの多重波抑制特性," 信学論(B-II)、Vol.J74-B-II, No.11, pp.599-607, Nov. (1991).

[2-4] T. Ohgane, T. Shimura, N. Matsuzawa and H. Sasaoka, "An implementation of a CMA adaptive array for high speed GMSK transmission in mobile communications," IEEE Trans. Veh. Technol., vol. 42, pp. 282-288, Aug. 1993.

[2-5] Y. Ogawa, Y. Nagashima and K. Itoh, "An Adaptive Antenna System for High-Speed Digital Mobile Communications," IEICE Trans. Commun., Vol. E75-B, No.5 May 1992.

[2-6]黒岩　登、河野隆二, "アダプティブアレーアンテナによる指向性ダイバーシチ受信の構成法," 信学論(B-II)、Vol.J73-B-II No.11, pp755-763, Nov. (1990).

[2-7] T. Ohgane, ﾒSpectral Efficiency Improvement by Base Station Antenna Pattern Control for Land Mobile Cellular Systems,ﾓ IEICE Trans. Commun., VOL E77-B, No. 5 May (1994).

[2-8] 千葉　勇、中條　渉、藤瀬　雅行: "ビームスペースCMAアダプティブアレーアンテナ,"信学論(B-II) Vol J77-B-II, No.3, pp.130-138, Mar. 1994.

[3-1] BURR-BROWN Japan, LTD., "Products Data Book 1991," pp 5-243- 5-253

[3-2] Analog Devices, "Data Converter Reference Manual Vol I 1992," pp 2-693- 2-708.

[4-1] VMEbus アーキテクチャ・マニュアル Revision C.3 VME MEMBER 1992年4月1日.

[4-2] VXIbus アーキテクチャ・マニュアル (VXIbus System Specification Revision 1.3) VME MEMBER 1990年4月8日.

[5-1] XILINX, "The Programmable Logic Data Book 1994, " 1994.

[6-1] 大滝幸夫, "移動体衛星通信用DBFアンテナ信号処理部の構成とその特性," ATRテクニカルレポート, TR-O-0046, Jun. 1992.

[6-2] 田中豊久, 三浦　龍, 千葉　勇, 唐沢好男、"ASICを用いたDBFマルチビームアンテナの開発," 信学論(B-II), Vol.J78-B-II, no.9, pp602-610, Sep. 1995.

第１０章　付録

第１節　DSP ASIC Bd 回路図

第１項　u1

sheet1　MAIN BLOCK
sheet2　FPGA I/O A/Dデータ入力(ch.A、ch.B)
sheet3　FPGA I/O FFTビームデータ出力(Ich、Qch)
sheet4　Local RAM/IIR FilteringとCh.A,Bの電力値比較部
sheet5　FFT 係数ROM(ch.A、ch.B)
sheet6　FPGA I/O ch.A、ch.B共通 X軸FFT-Ich
sheet7　FPGA I/O ch.A Y軸FFT-Ich
sheet8　FPGA I/O Master sequencer(u10)からのコントロール信号
sheet9　Processor ch.A
sheet10　Processor ch.B
sheet11　FPGA I/O ch.A、ch.B共通 X軸FFT-Qch
sheet12　FPGA I/O ch.A Y軸FFT-Qch
sheet13　FPGA I/O ch.B Y軸FFT-Ich
sheet14　FPGA I/O ch.B Y軸FFT-Qch
sheet15　選択チャンネルデコード
sheet16　FPGA I/O パワーデータ出力、FIRフィルタ係数入力
sheet17　FPGA I/O CPU I/F データバス
sheet18　CPU I/F(アドレスデコーダ、R/W、ライトパルス、リードイネーブル信号発生)
sheet19　CPU I/F(リードレジスタ、ライトレジスタ)
sheet20　FPGA I/O 内部信号モニタ用
sheet21　CPU I/F(リードレジスタ、ライトレジスタ)

第２項　u2～u8

FFT ROM
sheet1　FFT 係数ROM(u2, u3, u4)
sheet2　FFT 係数ROM(u5, u6)
sheet3　FFT 係数ROM(u7, u8)

第３項　u9

sheet1　Beam Selector
sheet3　FPGA I/O ビーム電力データ(u1、u2)
sheet4　FPGA I/O ビーム電力データ(u3、u4)
sheet5　FPGA I/O ビーム電力データ(u5、u6)
sheet6　FPGA I/O ビーム電力データ(u7、u8)
sheet8　FPGA I/O ビーム電力出力Control信号
sheet10　FPGA I/O Control信号、Beam最下位ビットアドレス、選択ビームアドレス
sheet11　FPGA I/O CPU I/F データバス
sheet12　CPU I/F(アドレスデコーダ、R/W、ライトパルス、リードイネーブル信号発生)

第４項　u10

sheet1　Clock management、Vector Rotator Section、Master Sequencer
sheet2　FPGA I/OFFTビームデータ Ich 入力
sheet3　FPGA I/OFFTビームデータ Qch 入力

sheet4 FIR係数ROM
sheet5 CPU I/F
sheet6 FPGA I/O CPU I/F データバス
sheet7 CPU I/F(リードレジスタ、ライトレジスタ)
sheet8 FPGA I/O CPU I/F、Clock、DSP出力
sheet9 FPGA I/O Control信号、選択ビームアドレス
sheet10 FPGA I/O FIR係数
sheet11 CPU I/F(リードレジスタ、ライトレジスタ)
sheet12 CPU I/F(リードレジスタ、ライトレジスタ)

第5項 Components
mux_3_2_1
mux84_1
mux82_1
comp_sel
sel_cont
shifter88
x74_374
ff
reg8
ram328
in_ofc
ffdc
mulb_reg
ss_u1p3
ms_u10
phc_rom
ram48
MC_RAM
ss_u10
proc8_3
cpu_cont
mux44_1
x74_374_2
reg8ofc
mux88_1
FDRS
ofc2
ss_comp4

第2節 Misc Information

第1項 ASIC Connection(internal)
第2項 Timing Chart
Multibeamseciton
Beam selection/Vector Rotation section

第 3 項　ROM Data

1. FFT ROM DATA

FFT ROM DATA

U1-Ch.0

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	0	1	1	1	1	1	1	1	1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	

U1-Ch.2

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	0	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	0	1	1	1	1	1	1	1	1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	0	1	1	1	1	1	1	1	1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	55	FFAA	FFAA	FFAA	FFAA	FFAA	FFAA	FFFF	

U2-Ch.4

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	1	0	0	0	0	0	0	1	-1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	1	0	0	0	0	0	0	1	-1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	3600	C9FF	C9FF	C9FF	C9FF	C9FF	C9FF	FFFF	

U2-Ch.6

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	0	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	0	1	1	1	1	1	1	1	1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	1	0	0	0	0	0	0	1	-1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	1	0	0	0	0	0	0	1	-1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	3655	C9AA	C9AA	C9AA	C9AA	C9AA	C9AA	FFFF	

U3-Ch.8

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	0	1	1	1	1	1	1	1	1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	0	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	5500	AAFF	AAFF	AAFF	AAFF	AAFF	AAFF	FFFF	

U3-Ch.10

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	0	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	0	1	1	1	1	1	1	1	1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	0	1	1	1	1	1	1	1	1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	0	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	5555	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	FFFF	

U4-Ch.12

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	1	0	0	0	0	0	0	1	-1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	1	0	0	0	0	0	0	1	-1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	6300	9CFF	9CFF	9CFF	9CFF	9CFF	9CFF	FFFF	

U4-Ch.14

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	0	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	0	1	1	1	1	1	1	1	1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	1	0	0	0	0	0	0	1	-1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	1	0	0	0	0	0	0	1	-1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	6355	9CAA	9CAA	9CAA	9CAA	9CAA	9CAA	FFFF	

U5-Ch.1

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	-1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	1	0	0	0	0	0	0	1	-1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	0	1	1	1	1	1	1	1	1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	36	FFC9	FFC9	FFC9	FFC9	FFC9	FFC9	FFFF	

U5-Ch.3

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	1	0	0	0	0	0	0	1	-1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	1	0	0	0	0	0	0	1	-1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	0	1	1	1	1	1	1	1	1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	63	FF9C	FF9C	FF9C	FF9C	FF9C	FF9C	FFFF	

U6-Ch.5

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	-1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	1	0	0	0	0	0	0	1	-1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	1	0	0	0	0	0	0	1	-1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	1	0	0	0	0	0	0	1	-1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	3636	C9C9	C9C9	C9C9	C9C9	C9C9	C9C9	FFFF	

U6-Ch.7

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	1	0	0	0	0	0	0	1	-1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	1	0	0	0	0	0	0	1	-1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	1	0	0	0	0	0	0	1	-1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	1	0	0	0	0	0	0	1	-1
14	0	1	1	1	1	1	1	1	1
15	0	1	1	1	1	1	1	1	1
ROM Data	3663	C99C	C99C	C99C	C99C	C99C	C99C	FFFF	

U7-Ch.9

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	-1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	1	0	0	0	0	0	0	1	-1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	0	1	1	1	1	1	1	1	1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	0	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	5536	AAC9	AAC9	AAC9	AAC9	AAC9	AAC9	FFFF	

U7-Ch.11

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	1	0	0	0	0	0	0	1	-1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	1	0	0	0	0	0	0	1	-1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	0	1	1	1	1	1	1	1	1
10	1	0	0	0	0	0	0	1	-1
11	0	1	1	1	1	1	1	1	1
12	1	0	0	0	0	0	0	1	-1
13	0	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	5563	AA9C	AA9C	AA9C	AA9C	AA9C	AA9C	FFFF	

U8-Ch.13

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	-1
2	1	0	0	0	0	0	0	1	-1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	-1
5	1	0	0	0	0	0	0	1	-1
6	0	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	1	0	0	0	0	0	0	1	-1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	1	0	0	0	0	0	0	1	-1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	6336	9CC9	9CC9	9CC9	9CC9	9CC9	9CC9	FFFF	

U8-Ch.15

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	1	0	0	0	0	0	0	1	-1
1	1	0	0	0	0	0	0	1	-1
2	0	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1
5	1	0	0	0	0	0	0	1	-1
6	1	0	0	0	0	0	0	1	-1
7	0	1	1	1	1	1	1	1	1
8	1	0	0	0	0	0	0	1	-1
9	1	0	0	0	0	0	0	1	-1
10	0	1	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1
12	0	1	1	1	1	1	1	1	1
13	1	0	0	0	0	0	0	1	-1
14	1	0	0	0	0	0	0	1	-1
15	0	1	1	1	1	1	1	1	1
ROM Data	6363	9C9C	9C9C	9C9C	9C9C	9C9C	9C9C	FFFF	

2. LOCAL ROM/FIR ROM/VECTOR ROTATION ROM

LOCAL COEFFICIENT SAMPLE

U1-Ch.0

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	0	-1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	-1
ROM Data	0090H	0009H	0009H	0009H	0009H	0009H	0009H	0009H	

FIR ROM COEFFICIENT

FIR ROM Roll off version

ADD/BIT	7	6	5	4	3	2	1	0	fix	係数	floating point
0	0	0	0	0	1	1	1	1	15		0.1142194
1	0	0	0	0	1	0	1	0	10		0.078306454
2	0	0	0	0	1	1	0	0	12		0.096336908
3	0	0	0	0	1	1	1	0	14		0.1097589
4	0	0	0	0	1	1	1	1	15		0.11681552
5	0	0	0	0	1	1	1	1	15		0.11681552
6	0	0	0	0	1	1	1	0	14		0.1097589
7	0	0	0	0	1	1	0	0	12		0.096336908
8	0	0	0	0	1	0	1	0	10		0.078306454
9	0	0	0	0	1	1	1	1	15		0.1142194
10	0	0	0	0	0	0	0	0	0		0
11	0	0	0	0	0	0	0	0	0		0
12	0	0	0	0	0	0	0	0	0		0
13	0	0	0	0	0	0	0	0	0		0
14	0	0	0	0	0	0	0	0	0		0
15	0	0	0	0	0	0	0	0	0		0
ROM Data	0000H	0000H	0000H	0000H	03FFH	02FDH	037BH	0231H			

FIR ROM Root Roll off version

ADD/BIT	7	6	5	4	3	2	1	0	fix	係数	floating point
0	1	1	1	1	1	0	1	1	-5		-0.04065529
1	0	0	0	0	0	1	0	1	5		0.041872471
2	0	0	0	0	1	1	1	0	14		0.11086148
3	0	0	0	1	1	0	0	0	24		0.19127146
4	0	0	0	1	1	1	1	1	31		0.24484269
5	0	0	0	1	1	1	1	1	31		0.24484269
6	0	0	0	1	1	0	0	0	24		0.19127146
7	0	0	0	0	1	1	1	0	14		0.11086148
8	0	0	0	0	0	1	0	1	5		0.041872471
9	1	1	1	1	1	0	1	1	-5		-0.04065529
10	0	0	0	0	0	0	0	0	0		0
11	0	0	0	0	0	0	0	0	0		0
12	0	0	0	0	0	0	0	0	0		0
13	0	0	0	0	0	0	0	0	0		0
14	0	0	0	0	0	0	0	0	0		0
15	0	0	0	0	0	0	0	0	0		0
ROM Data	0201H	0201H	0201H	0279H	02FDH	01B6H	02B5H	0333H			

PHSE SHIFTER ROM DATA

REAL

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	1	1	1	1	1	1	1	127
1	1	0	1	0	0	1	1	0	-90
2	0	0	0	0	0	0	0	0	0
3	1	0	1	0	0	1	1	0	-90
4	1	0	1	0	0	1	1	0	-90
5	0	0	0	0	0	0	0	0	0
6	0	1	0	1	1	0	1	0	90
7	0	1	1	1	1	1	1	1	127
8	0	0	0	0	0	0	0	0	0
9	0	1	0	1	1	0	1	0	90
10	1	0	0	0	0	0	0	0	-128
11	1	0	1	0	0	1	1	0	-90
12	1	0	1	0	0	1	1	0	-90
13	0	1	1	1	1	1	1	1	127
14	1	0	1	0	0	1	1	0	-90
15	0	0	0	0	0	0	0	0	0
ROM Data	5C1A	22C1	789B	22C1	22C1	789B	7ADB	2081	

IMGINARY

ADD/BIT	7	6	5	4	3	2	1	0	係数
0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	0	-90
2	0	1	1	1	1	1	1	1	127
3	0	1	0	1	1	0	1	0	90
4	1	0	1	0	0	1	1	0	-90
5	0	1	1	1	1	1	1	1	127
6	1	0	1	0	0	1	1	0	-90
7	0	0	0	0	0	0	0	0	0
8	0	1	1	1	1	1	1	1	127
9	1	0	1	0	0	1	1	0	-90
10	0	0	0	0	0	0	0	0	0
11	1	0	1	0	0	1	1	0	-90
12	0	1	0	1	1	0	1	0	90
13	0	0	0	0	0	0	0	0	0
14	1	0	1	0	0	1	1	0	-90
15	1	0	0	0	0	0	0	0	-128
ROM Data	CA52	112C	4B76	112C	112C	4B76	5B7E	124	

第 4 項　CPU Control Sample program

1. A16 ModeでのRead Sample Program

```
#include <stdio.h>
#include <sicl.h>

static vmeinit();
static void int_setup();
static void int_hndlr();
int intr = 0;
int input_data;
char first_ch;
unsigned short out_data;

main()
{
    int o;
    INST id_intf1;
    unsigned long mask = 1;
    int laddr = 1;
    char str[64];

    ionerror(I_ERROR_EXIT);
    sprintf(str,"vxi,%d",laddr);
    id_intf1 = iopen(str);
    int_setup(id_intf1,mask);

    read_register(id_intf1,laddr);
    iclose(id_intf1);
}

static void int_setup(id,mask)
INST id;
unsigned long mask;
{
    ionintr(id,int_hndlr);
    isetintr(id,I_INTR_VXI_SIGNAL,mask);
}


static void int_hndlr(id,reason,sec)
INST id;
long reason;
long sec;
{
    printf("VME interrupt: reason: 0x%x, sec: 0x%x\n",reason,sec);
    intr = 1;
```

```
}

read_register(id,laddr)
INST id;
unsigned short laddr;
{
      unsigned short *ptr;
      unsigned short data;
      int reg=0;
      char *a16_ptr = 0;

      a16_ptr = imap(id,I_MAP_A16,0,1,0);
      ptr = (unsigned short *)(a16_ptr + 0xc000 + laddr *64 + reg);

      data = *ptr++;
      printf(" Reg addr #0 - %x \n",data);
      data = *ptr++;
      printf(" Reg addr #2 - %x \n",data);
      data = *ptr++;
      printf(" Reg addr #4 - %x \n",data);
      data = *ptr++;
      printf(" Reg addr #6 - %x \n",data);

}
```

2. A16 Modeでの Write Sample Program

```
#include <stdio.h>
#include <sicl.h>

static vmeinit();
static void int_setup();
static void int_hndlr();
int intr = 0;

main()
{
      int o;
      INST id_intf1;
      unsigned long mask = 1;
      int laddr = 1;
      char str[64];

      ionerror(I_ERROR_EXIT);
      sprintf(str,"vxi,%d",laddr);
      id_intf1 = iopen(str);
      int_setup(id_intf1,mask);
```

```
    write_register(id_intf1,laddr);
    iclose(id_intf1);
}

static void int_setup(id,mask)
INST id;
unsigned long mask;
{
    ionintr(id,int_hndlr);
    isetintr(id,I_INTR_VXI_SIGNAL,mask);
}

write_register(id,laddr)
INST id;
unsigned short laddr;
{
    int reg = 4;
    char *a16_ptr;
    a16_ptr = imap(id,I_MAP_A16,0,1,0);
    *(unsigned short *)(a16_ptr + 0xc000 + laddr *64 + reg) = 0x01;
}

static void int_hndlr(id,reason,sec)
INST id;
long reason;
long sec;
{
    printf("VME interrupt: reason: 0x%x, sec: 0x%x\n",reason,sec);
    intr = 1;
}
```

3. A24 ModeでのRead Sample Program

```
#include <stdio.h>
#include <sicl.h>

static vmeinit();
static void int_setup();
static void int_hndlr();
int intr = 0;
unsigned short rdata;

main()
{
    int o;
    INST id_intf1;
    unsigned long mask = 1;
    int laddr = 1;
```

```
    char str[64];

    ionerror(I_ERROR_EXIT);
    sprintf(str,"vxi,%d",laddr);
    id_intf1 = iopen(str);
    int_setup(id_intf1,mask);

    read_register24(id_intf1,laddr);

    iclose(id_intf1);
}

static void int_setup(id,mask)
INST id;
unsigned long mask;
{
    ionintr(id,int_hndlr);
    isetintr(id,I_INTR_VXI_SIGNAL,mask);
}

static void int_hndlr(id,reason,sec)
INST id;
long reason;
long sec;
{
    printf("VME interrupt: reason: 0x%x, sec: 0x%x\n",reason,sec);
    intr = 1;
}

read_register24(id,laddr)
INST id;
unsigned short laddr;
{
    unsigned short *ptr;
    unsigned short *ptr24;
    unsigned short *a24_ptr;
    unsigned short reg, offset;
    char *a16_ptr = 0;
      int i,j;

    a16_ptr = imap(id,I_MAP_A16,0,1,0);
    ptr = (unsigned short *)(a16_ptr + 0xc000 + laddr *64 + 6);
    offset = *ptr;
    printf(" Reg addr #6 - %x \n",offset);

      a24_ptr = (unsigned short *)imap(id, I_MAP_A24, (offset >>8), 0x01, 0);

        /* Access U1 A24 mode */
```

```
        /* PRGRAMMABLE MODE */
        /* Each IC is allocated by offset value of 16. */
        /* U1: 0, U2:16, U3:32, U4:48, U5:64, U6:80, U7:96, U8:112 */

    reg = 16+1;
    ptr24 = (unsigned short *)(a24_ptr + reg);
    rdata = *ptr24++;

}
```

第 3 節　VHDL Listing

第1項　ms_u10 VHDL Listing

```
--
-- Copyright(c) 1993, 1994, 1995, and 1996 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Mon, December, 20, 1993
--
-- Funtion
--
-- STATE GENERATOR with MCLK/MCLK2
--
-- Main Sequencer controls the following signals.
--
-- OUT_EN(0): IQ RAM WRITE ENABLE
-- OUT_EN(1): SPECTRUM SELECTOR
-- OUT_EN(2): ACC ENABLE
--
-- OUT_TRIG(0): Beam Form Ich OUTPUT TRIG : 0~21
-- OUT_TRIG(1): Beam Form Qch OUTPUT TRIG : 0~21
-- OUT_TRIG(2): FIR Ich OUTPUT TRIG : 0~21
-- OUT_TRIG(3): FIR Qch OUTPUT TRIG : 0~21
-- OUT_TRIG(4): FFT 1st REAL OUTPUT TRIG : 22~31
-- OUT_TRIG(5): FIR 1st IMAG. OUTPUT TRIG : 22~31
-- OUT_TRIG(6): SPEC OUTPUT TRIG    : 47~48
-- OUT_TRIG(7): IQ TRIG

-- OUT_TRIG(7): ACC ENABLE
-- OUT_TRIG(8): WRITE ENABLE
--
-- FILTER COEFFICIENT DATA ROM ADDRESS GENERATION
-- MRST and START are active low signal in this module.

-- OUT_TRIGs are trigged by MCLK2 3/8/1994
-- SPEC_SEL is trigged by MCLK 3/8/1994

library mgc_portable;
use mgc_portable.qsim_logic.all;
```

```
-- MAIN Sequencer Entity Description

entity ms_u10 is
  port(
    MCLK,MRST,START: in qsim_state;
    OUT_TRIG: out qsim_state_vector(7 downto 0) := (others=> '0');
    OUT_EN: out qsim_state_vector(2 downto 0) := (others => '0');
    FIR_STATE,FFT1_STATE,FFT2_STATE,SPEC_STATE: out qsim_state := '0';
    FIR_ADD: out qsim_state_vector(3 downto 0) := "0000"
  );
end ms_u10;

-- master_seq Architecture Description
-- Moore Machine Design Model

architecture rtl of ms_u10 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal istate : integer range 0 to 63;
  signal count, count2 : integer range 0 to 63;
  signal fir_en, fft1_en, fft2_en, spec_en : qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;
  signal pre_fir_en, pre_fft1_en, pre_fft2_en, pre_spec_en : qsim_state;
  signal start2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type  fir_trig_states IS (s00, s01, s02);
  type  fft1_trig_states IS (s10, s11, s12);
  type  fft2_trig_states IS (s20, s21, s22);
  type  spec_trig_states IS (s30, s31);
  type  fir_iq_states IS (s50, s51);

  signal present_fir_state : fir_trig_states := s00;
  signal next_fir_state   : fir_trig_states := s01;
  signal present_fft1_state : fft1_trig_states := s10;
  signal next_fft1_state   : fft1_trig_states := s11;
  signal present_fft2_state : fft2_trig_states := s20;
  signal next_fft2_state   : fft2_trig_states := s21;
  signal present_spec_state : spec_trig_states := s30;
  signal next_spec_state   : spec_trig_states := s31;
  signal present_iq_trig_state : fir_iq_states := s50;
  signal next_iq_trig_state   : fir_iq_states := s51;
```

```
  signal fir_count: integer range 0 to 27 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 18 := 0;
  signal TRIGGER:  qsim_state_vector(7 downto 0);

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
  type   fir_en_states IS (ss00, ss01);
  type   spec_en_states IS (ss10, ss11);
  type   acc_en_states IS (ss20, ss21);

  signal present_fir_state2 : fir_en_states := ss00;
  signal next_fir_state2    : fir_en_states := ss01;
  signal present_spec_state2 : spec_en_states := ss10;
  signal next_spec_state2    : spec_en_states := ss11;
  signal present_acc_state : acc_en_states := ss20;
  signal next_acc_state    : acc_en_states := ss21;

  signal fir_count2: integer range 0 to 27 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 18 := 0;
  signal ENABLE: qsim_state_vector(2 downto 0);
  signal fir_count_en: qsim_state;
  signal fir_current_add: qsim_state_vector(3 downto 0):= (OTHERS=>'0');

begin

-------------------------------------------------------------------
-------------------------------------------------------------------
--       CONTORL SEQUENCE SUMMARY
--
-- Parameter ITEM          EN*          TRIG
-- OUTPUT_xxx     Final OUTPUT       MCLK2              MCLK
-- xxx_count STATE FIX   MCLK                 MCLK2
-- xxx_count each COUNT         MCLK2              MCLK
-- xxx_en      Phase Change      MCLK               MCLK2
-- countx      COUNT             MCLK2              MCLK

 START2_PROCESS:process(MRST, MCLK)
  begin
   if ( MRST = '1') then
     start2 <= '1';
   elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
```

```
    start2 <= START;
   end if;
 end process START2_PROCESS;

 INCREMENT_PROCESS:process(istate)
 begin
  count <= istate;
 end process INCREMENT_PROCESS;

 INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK.
 begin
 if ( START = '0' ) then -- Initializing local phase counters
  istate <= 0;
  fir_count <= 0;
  fft1_count <= 0;
  fft2_count <= 0;
  spec_count <= 0;

  fir_en2 <= '0';
  fft1_en2 <= '0';
  fft2_en2 <= '0';
  spec_en2 <= '0';

 elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
  istate <= count +1;
  fir_en2 <= fir_en;
  fft1_en2 <= fft1_en;
  fft2_en2 <= fft2_en;
  spec_en2 <= spec_en;

-- The followings are counter for EN*
  if (fir_en = '1') then
   fir_count <= fir_count + 1;
  end if;
  if (fft1_en = '1') then
   fft1_count <= fft1_count + 1;
  end if;
  if (fft2_en = '1') then
   fft2_count <= fft2_count + 1;
  end if;
  if (spec_en = '1') then
   spec_count <= spec_count + 1;
  end if;
 end if;
 end process INIT_SEQ;

-- STATE for TRIG must be tirggered by MCLK2.
-- count2 are delayed 1/2 clock cycle.
```

```
INIT_SEQ2:process(START2,MCLK)
begin
if ( START2 = '0' ) then
   count2 <= 0;
   fir_count2 <= 0;
   fft1_count2 <= 0;
   fft2_count2 <= 0;
   spec_count2 <= 0;

elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
   count2 <= count;

   fir_count2 <=  fir_count;
   fft1_count2 <= fft1_count;
   fft2_count2 <= fft2_count;
   spec_count2 <= spec_count;


end if;
end process INIT_SEQ2;


State_Process: process(count)
begin
  if ( count < 23 ) then -- less than 23
    pre_fir_en  <= '1';
    pre_fft1_en <= '0';
    pre_fft2_en <= '0';
    pre_spec_en <= '0';
  elsif ( count < 27 ) then -- less than 27
    pre_fir_en  <= '1';
    pre_fft1_en <= '1';
    pre_fft2_en <= '0';
    pre_spec_en <= '0';
  elsif ( count < 34 ) then -- less than 34
    pre_fir_en  <= '0';
    pre_fft1_en <= '1';
    pre_fft2_en <= '0';
    pre_spec_en <= '0';
  elsif ( count < 39 ) then -- less than 39
    pre_fir_en  <= '0';
    pre_fft1_en <= '1';
    pre_fft2_en <= '1';
    pre_spec_en <= '0';
  elsif ( count < 45 ) then -- less than 45
    pre_fir_en  <= '0';
    pre_fft1_en <= '0';
```

```
      pre_fft2_en <= '1';
      pre_spec_en <= '0';
     elsif ( count < 49 ) then -- less than 49
      pre_fir_en  <= '0';
      pre_fft1_en <= '0';
      pre_fft2_en <= '1';
      pre_spec_en <= '1';
     elsif ( count <= 64 ) then  -- less than 64
      pre_fir_en  <= '0';
      pre_fft1_en <= '0';
      pre_fft2_en <= '0';
      pre_spec_en <= '1';
     else                    -- more than 64
      pre_fir_en  <= '0';
      pre_fft1_en <= '0';
      pre_fft2_en <= '0';
      pre_spec_en <= '0';
     end if;
  end process State_Process;

  STATE_LATCH_PROCESS: process(START, MCLK)
  begin
  if ( START = '0') then
   fir_en <= '1';
   fft1_en <= '0';
   fft2_en <= '0';
   spec_en <= '0';
   FIR_STATE <= '1';
   FFT1_STATE <= '0';
   FFT2_STATE <= '0';
   SPEC_STATE <= '0';
  elsif ( MCLK = '1' and MCLK'event and MCLK'last_value = '0' ) then
   fir_en <= pre_fir_en;
   FIR_STATE <= pre_fir_en;
   fft1_en <= pre_fft1_en;
   FFT1_STATE <= pre_fft1_en;
   fft2_en <= pre_fft2_en;
   FFT2_STATE <= pre_fft2_en;
   spec_en <= pre_spec_en;
   SPEC_STATE <= pre_spec_en;
  end if;
  end process STATE_LATCH_PROCESS;

  ------------------------------------------------------------------------
  -- TRIG SEQUENCE TRIGGERD by MCLK2
  ------------------------------------------------------------------------

  IQ_TRIG_STATE_DECODE:process(fir_count)
```

```
 begin
 next_iq_trig_state <= s50;
 case fir_count is
  when 3 => next_iq_trig_state <= s51;  -- ****************************
  when 14 => next_iq_trig_state <= s51;  -- ****************************
  when OTHERS => next_iq_trig_state <= s50;
 end case;
 end process IQ_TRIG_STATE_DECODE;

-------------------------------------------------------------------------

 FIR_TRIG_STATE_DECODE:process(fir_count)
 begin
 next_fir_state <= s00;
 case fir_count is
  when 13 => next_fir_state <= s01;  -- ****************************
  when 24 => next_fir_state <= s02;  -- ****************************
  when OTHERS => next_fir_state <= s00;
 end case;
 end process FIR_TRIG_STATE_DECODE;

-------------------------------------------------------------------------

 FFT1_TRIG_STATE_DECODE:process(fft1_count)
 begin
 next_fft1_state <= s10;
 case fft1_count is
  when 7 => next_fft1_state <= s11;  -- ****************************
  when 12 => next_fft1_state <= s12;  -- ****************************
  when OTHERS => next_fft1_state <= s10;
 end case;
 end process FFT1_TRIG_STATE_DECODE;

-------------------------------------------------------------------------

 FFT2_TRIG_STATE_DECODE:process(fft2_count)
 begin
 next_fft2_state <= s20;
 case fft2_count is
  when 7 => next_fft2_state <= s21;  -- ****************************
  when 12 => next_fft2_state <= s22;  -- ****************************
  when OTHERS => next_fft2_state <= s20;
 end case;
 end process FFT2_TRIG_STATE_DECODE;

-------------------------------------------------------------------------

 SPEC_TRIG_STATE_DECODE:process(spec_count)
 begin
```

```
next_spec_state <= s30;
case spec_count is
 when 4 => next_spec_state <= s31;  -- ****************************
 when OTHERS => next_spec_state <= s30;
end case;
end process SPEC_TRIG_STATE_DECODE;

-------------------------------------------------------------------------------
TRIG_STATE_REGISTER:process(MCLK, START2)

begin
 if (START2 = '0') then
     present_fir_state <= s00;
     present_iq_trig_state <= s50;
     present_fft1_state <= s10;
     present_fft2_state <= s20;
     present_spec_state <= s30;
 elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
       if ( FIR_EN2 = '1' ) then
         present_fir_state <= next_fir_state;
         present_iq_trig_state <= next_iq_trig_state;
    end if;
       if ( FFT1_EN2 = '1' ) then
         present_fft1_state <= next_fft1_state;
    end if;
       if ( FFT2_EN2 = '1' ) then
         present_fft2_state <= next_fft2_state;
    end if;
       if ( SPEC_EN2 = '1' ) then
         present_spec_state <= next_spec_state;
       end if;
 end if;
end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------------

IQ_REG_TRIG_DECODE:process(present_iq_trig_state)
begin
case present_iq_trig_state is
 when s50 => TRIGGER(7) <= '0'; -- Trig nothing
 when s51 => TRIGGER(7) <= '1'; -- Trig WE
end case;
end process IQ_REG_TRIG_DECODE;

-------------------------------------------------------------------------------

FIR_OUTPUT_TRIG_DECODE:process(present_fir_state)
begin
```

```
 case present_fir_state is
  when s00 => TRIGGER(1 downto 0) <= "00"; -- Trig nothing
  when s01 => TRIGGER(1 downto 0) <= "01"; -- Trig Beam Form Ich
  when s02 => TRIGGER(1 downto 0) <= "10"; -- Trig Beam Form Qch
 end case;
 end process FIR_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------

 FFT1_OUTPUT_TRIG_DECODE:process(present_fft1_state)
 begin
 case present_fft1_state is
  when s10 => TRIGGER(3 downto 2) <= "00"; -- Trig nothing
  when s11 => TRIGGER(3 downto 2) <= "01"; -- Trig FFT 1st REAL
  when s12 => TRIGGER(3 downto 2) <= "10"; -- Trig FFT 1st IMAG.
 end case;
 end process FFT1_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------

 FFT2_OUTPUT_TRIG_DECODE:process(present_fft2_state)
 begin
 case present_fft2_state is
  when s20 => TRIGGER(5 downto 4) <= "00"; -- Trig nothing
  when s21 => TRIGGER(5 downto 4) <= "01"; -- Trig FFT 1st REAL
  when s22 => TRIGGER(5 downto 4) <= "10"; -- Trig FFT 1st IMAG.
 end case;
 end process FFT2_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
 SPEC_OUTPUT_TRIG_DECODE:process(present_spec_state)
 begin
 case present_spec_state is
  when s30 => TRIGGER(6) <= '0';
  when s31 => TRIGGER(6) <= '1';
 end case;
 end process SPEC_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, START)
 begin
 if ( START = '0' ) then
   OUT_TRIG <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
   OUT_TRIG <= TRIGGER;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;
```

```
-------------------------------------------------------------------------
-- ENABLE CONTROL TRIGGERED by MCLK
-- This control includes WE, SPEC_SEL, ACC_EN
-------------------------------------------------------------------------

 IQ_WE_STATE_DECODE:process(fir_count2)
 begin
 next_fir_state2 <= ss00;
 case fir_count2 is
  when 11 => next_fir_state2 <= ss01; -- ************************************
  when 22 => next_fir_state2 <= ss01; -- ************************************
  when OTHERS => next_fir_state2 <= ss00;
 end case;
 end process IQ_WE_STATE_DECODE;

-------------------------------------------------------------------------


 SPEC_SEL_STATE_DECODE:process(fft2_count2)
 begin
 next_spec_state2 <= ss10;
 case fft2_count2 is
  when 11 => next_spec_state2 <= ss11; -- ************************************
  when OTHERS => next_spec_state2 <= ss10;
 end case;
 end process SPEC_SEL_STATE_DECODE;

-------------------------------------------------------------------------

 ACC_EN_STATE_DECODE:process(count2)
 begin
 next_acc_state <= ss20;
 case count2 is
  when 0 | 1 | 2 => next_acc_state <= ss21;  -- ****************************
  when 13 => next_acc_state <= ss21;  -- ****************************
  when 24 | 25 => next_acc_state <= ss21;  -- ****************************
  when 30 => next_acc_state <= ss21;  -- ****************************
  when 35 | 36 => next_acc_state <= ss21;  -- ****************************
  when 41 => next_acc_state <= ss21;  -- ****************************
  when 46 => next_acc_state <= ss21;  -- ****************************
  when 49 | 50 => next_acc_state <= ss21;  -- ****************************
  when OTHERS => next_acc_state <= ss20;
 end case;
 end process ACC_EN_STATE_DECODE;

-------------------------------------------------------------------------
```

```
 EN_STATE_REGISTER:process(MCLK, START)

 begin
  if (START = '0') then
      present_fir_state2 <= ss00;
      present_spec_state2 <= ss10;
        present_acc_state <= ss21;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        if ( FIR_EN = '1' ) then
          present_fir_state2 <= next_fir_state2;
          present_acc_state <= next_acc_state;
      end if;
        if ( FFT1_EN = '1' ) then
          present_acc_state <= next_acc_state;
      end if;
        if ( FFT2_EN = '1' ) then
          present_acc_state <= next_acc_state;
          present_spec_state2 <= next_spec_state2;
      end if;
        if ( SPEC_EN = '1' ) then
          present_acc_state <= next_acc_state;
        end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------------

 FIR_IQ_WE_DECODE:process(present_fir_state2)
 begin
 case present_fir_state2 is
  when ss00 => ENABLE(0) <= '0'; -- READ
  when ss01 => ENABLE(0) <= '1'; -- WRITE
 end case;
 end process FIR_IQ_WE_DECODE;

-------------------------------------------------------------------------------
 SPEC_OUTPUT_SEL_DECODE:process(present_spec_state2)
 begin
 case present_spec_state2 is
  when ss10 => ENABLE(1) <= '0';
  when ss11 => ENABLE(1) <= '1';
 end case;
 end process SPEC_OUTPUT_SEL_DECODE;
-------------------------------------------------------------------------------

 ACC_EN_TRIG_DECODE:process(present_acc_state)
 begin
 case present_acc_state is
```

```
  when ss20 => ENABLE(2) <= '0'; -- ACC ENABLE
  when ss21 => ENABLE(2) <= '1'; -- ACC RESET
 end case;
 end process ACC_EN_TRIG_DECODE;

-------------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START2)
 begin
 if ( START2 = '0' ) then
   OUT_EN <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   OUT_EN <= ENABLE;
 end if;

 end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------------
-- FIR Coef ROM CONTROL PROCESSES
-------------------------------------------------------------------------------

 FIR_WN_EN_PROCESS:process(START,fir_count )
 begin
  if (START = '0') then
     fir_count_en <= '0';
  elsif (fir_count < 3 ) then
     fir_count_en <= '0';
  elsif (fir_count < 13 ) then
     fir_count_en <= '1';
  elsif (fir_count < 14 ) then
     fir_count_en <= '0';
  elsif (fir_count < 25 ) then
     fir_count_en <= '1';
  else
     fir_count_en <= '0';
  end if;
 end process FIR_WN_EN_PROCESS;

 FIR_INC_CLK_PROCESS:process(MCLK, START2)
  begin
  if (START2 = '0') then
   fir_current_add <= (OTHERS => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
     if (fir_count_en = '1' ) then
         if (fir_current_add = "1001") then
           fir_current_add <= "0000";
       else
        fir_current_add <= fir_current_add + "0001";
```

```
      end if;
    end if;
  end if;
 end process FIR_INC_CLK_PROCESS;

 FIR_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        FIR_ADD <= fir_current_add;
    end if;
 end process FIR_ADD_CLK;
-------------------------------------------------------------------------
end rtl;
```

第2項 ss_u1 のVHDL Listing

```
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.
-- MCLK2 has been eliminated due to minimize clock skew.
-- OUT_EN and FIR_SEL are proceeded 1/2 MCLK, because of timing optimization.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u1p3 is
  port(
```

```
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state; -- eliminated MCLK2
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u1p3;

-- sseq_u1p3 Architecture Description

architecture rtl of ss_u1p3 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state := '1';
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
```

```
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;

-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
  when 1 => next_rd_state <= s31;  -- ****************************
  when 3 => next_rd_state <= s32;  -- ****************************
```

```
  when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;

 RD_EN_DECODE:process(present_rd_state)
 begin
  case present_rd_state is
   when s31 => rd_enable <= "101"; -- Enable ch.A
   when s32 => rd_enable <= "110"; -- Enable ch.B
   when others => rd_enable <= "000"; -- Enable nothing
  end case;
 end process RD_EN_DECODE;

 RD_EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
        present_rd_state <= s30;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
  end if;
 end process RD_EN_STATE_REGISTER;

 RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   RD_CONT <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   RD_CONT <= rd_enable;
 end if;
 end process RD_OUTPUT_REGISTER_TRIGGER;

 ------------------------------------------------------------------------
 -- END OF RAW DATA MONITOR ENABLE CONTROL
 ------------------------------------------------------------------------

  START2_PROCESS:process(MRST, MCLK)
  begin
   if ( MRST = '1') then
        start2 <= '1';
   elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        start2 <= START;
   end if;
  end process START2_PROCESS;

  INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
   begin
    if ( START = '0' ) then -- Initializing local phase counters
      fir_count <= 0;
```

```
        fft1_count <= 0;
      fft2_count <= 0;
        spec_count <= 0;
      fir_en2 <= '0';
      fft1_en2 <= '0';
      fft2_en2 <= '0';
      spec_en2 <= '0';
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
      fir_en2 <= fir_en;
      fft1_en2 <= fft1_en;
      fft2_en2 <= fft2_en;
      spec_en2 <= spec_en;
-- The followings are counter for EN*
     if (fir_en = '1') then
      fir_count <= fir_count + 1;
     end if;
     if (fft1_en = '1') then
      fft1_count <= fft1_count + 1;
     end if;
     if (fft2_en = '1') then
      fft2_count <= fft2_count + 1;
     end if;
     if (spec_en = '1') then
      spec_count <= spec_count + 1;
     end if;
    end if;
  end process INIT_SEQ;

 INIT_SEQ2:process(start2,MCLK)
 begin
 if ( start2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

 elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
 end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
```

```
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------------

 FFT11_EN_STATE_DECODE:process(fft1_count2)
 begin
 next_fft11_state2 <= ss10;
 case fft1_count2 is
  when 1 => next_fft11_state2 <= ss11;  -- ****************************
  when 3 => next_fft11_state2 <= ss11;  -- ****************************
  when 6 => next_fft11_state2 <= ss11;  -- ****************************
  when 8 => next_fft11_state2 <= ss11;  -- ****************************
  when OTHERS => next_fft11_state2 <= ss10;
 end case;
 end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------------
 FFT21_EN_STATE_DECODE:process(fft2_count2)
 begin
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 1 => next_fft21_state2 <= ss21;  -- ****************************
  when 6 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        if ( fft1_en = '1' ) then
          present_fft11_state2 <= next_fft11_state2;
        end if;
        if ( fft2_en = '1' ) then
          present_fft21_state2 <= next_fft21_state2;
        end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
```

```
  case present_fft11_state2 is
   when ss10 => ENABLE(0) <= '1'; -- Enable nothing
   when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
   when others => ENABLE(0) <= '1'; -- Enable nothing
  end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
   when 1 => next_fft21_state <= s11;  -- ***************************
   when 6 => next_fft21_state <= s11;  -- ***************************
   when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
        if ( FFT1_EN2 = '1' ) then
          present_fft21_state <= next_fft21_state;
        end if;
  end if;
 end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------
```

```
 FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.B
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.A
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

---------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
---------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START) -- Triggered by MCLK2
 begin
 if ( START = '0' ) then
   FFT_SEL <= ( OTHERS => '0');
   INPUT_DATA_SEL <= ( OTHERS => '0');
   IQ_RAW_SEL <= '0';
   FIR_SEL <= '0';
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
   FFT_SEL <= mux_sel2;
   INPUT_DATA_SEL <= mux_sel;
   IQ_RAW_SEL <= mux_sel3;
   FIR_SEL <= OUTSEL;
 end if;
 end process OUTPUT_REGISTER_EN;

---------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   OUT_EN <= ( OTHERS => '1');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   OUT_EN <= ENABLE;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;

---------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
---------------------------------------------------------------------------

 INPUT_DATA_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (spec_en = '1' ) then
```

```
    mux_sel <= "11";
   elsif ( fft2_en = '1') then
    mux_sel <= "10";
   elsif ( fft1_en = '1') then
    mux_sel <= "10";
   else
    case fir_count is
     when 1 => mux_sel <= "00"; --*********************************
     when 12 => mux_sel <= "00"; --*********************************
     when others => mux_sel <= "01";
    end case;
   end if;
 end if;
end process INPUT_DATA_MUX;

-------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------

 IQ_RAW_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel3 <= '0';
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fir_en = '1') then
    case fir_count is
     when 11 => mux_sel3 <= '1';
     when 22 => mux_sel3 <= '1';
     when others => mux_sel3 <= '0';
    end case;
   end if;
  end if;
 end process IQ_RAW_MUX;

-------------------------------------------------------------------------
-- FFT MUX PROCESSES ************* Each Sequencer has own procedure ********
-------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft2_en = '1') then
    if (fft2_count < 6) then --*********************************
     mux_sel2 <= "10";
    elsif (fft2_count < 11) then --*********************************
     mux_sel2 <= "11";
```

```
    end if;
   elsif ( fft1_en = '1') then --**********************************
    if (fft1_count < 6) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 11) then --**********************************
     mux_sel2 <= "01";
    end if;
   end if;
  end if;
 end process FFT_MUX;

 ---------------------------------------------------------------------------
 -- FFT Wn RAM CONTROL PROCESSES
 ---------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
     fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fft_en = '1') then
    if    (fft1_count2 < 2 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft1_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 11 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 2) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 11 ) then  --**********************************
     fft_count_en <= '1';
    else
     fft_count_en <= '0';
    end if;
   end if;
  end if;
 end process FFT_WN_EN_PROCESS;
```

```
 FFT_INC_CLK_PROCESS:process(MCLK, START)
 begin
 if (START = '0') then
  fft_current_add <= (OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fft_count_en = '1' ) then
       fft_current_add <= fft_current_add + "0001";
    end if;
 end if;
 end process FFT_INC_CLK_PROCESS;

 FFT_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        FFT_WN_ADD <= fft_current_add;
   end if;
 end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 ADD_STORE_EN_PROCESS:process(fir_count2)
 begin
 if (fir_count2 = 3) then
    add_store_en <= '1';
 else
    add_store_en <= '0';
 end if;
 end process ADD_STORE_EN_PROCESS;

 RD_RAM_LD_PROCESS:process(MRST, MCLK)
 begin
 if (MRST = '1') then
  ram_initial_add <= (others => '0');
 elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
  if ( add_store_en = '1') then
   ram_initial_add <= rd_current_add(3 downto 0);
  end if;
 end if;
 end process RD_RAM_LD_PROCESS;

 RD_RAM_EN_PROCESS:process(MCLK)
 begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en2 = '1') then
   if (fir_count2 < 2 ) then
    ram_count_en <= '0';
    iq_state <= '0';
```

```
    elsif (fir_count2 < 11) then
     ram_count_en <= '1';
     iq_state <= '0';
    elsif (fir_count2 = 11) then
     ram_count_en <= '0';
     iq_state <= '0';
    elsif (fir_count2 < 22 ) then
     ram_count_en <= '1';
     iq_state <= '1';
    else
     ram_count_en <= '0';
     iq_state <= '0';
    end if;
   end if;
  end if;
 end process RD_RAM_EN_PROCESS;

 RD_RAM_INC_CLK_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (start2 = '0') then
      rd_current_add(3 downto 0) <= ram_initial_add;
      rd_current_add(4) <= '0';
    elsif (ram_count_en = '1' ) then
     if ( rd_current_add(3 downto 0) = "1001") then
         rd_current_add(3 downto 0) <= "0000";
         rd_current_add(4) <= iq_state;
     else
       rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
         rd_current_add(4) <= iq_state;
     end if;
    end if;
  end if;
 end process RD_RAM_INC_CLK_PROCESS;

 RD_RAM_ADD_OUT_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   RD_RAM_ADD <= rd_current_add;
  end if;
 end process RD_RAM_ADD_OUT_PROCESS;

-------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------

 LOCAL_WN_EN_PROCESS:process(MCLK)
 begin
```

```
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
    if (fir_en = '1') then
      case fir_count is
        when 4 => local_count_en <= '1'; --*********************************
        when 15 => local_count_en <= '1'; --*********************************
        when others => local_count_en <= '0';
      end case;
    end if;
  end if;
end process LOCAL_WN_EN_PROCESS;

  LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
  begin
  if ( PHASE_CLK = '0' ) then
     local_current_add <= ( OTHERS => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
     if (local_count_en = '1' ) then
       if ( local_current_add = "111" ) then
          local_current_add <= "000";
       else
        local_current_add <= local_current_add + "001";
       end if;
     end if;
  end if;
  end process LOCAL_INC_CLK_PROCESS;

  LOCAL_WN_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
         LOCAL_WN_ADD <= local_current_add;
    end if;
  end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------------
end rtl;
```

第3項　ss_u2 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
```

```
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u2p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START, MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u2p3;

-- ss_u2p3 Architecture Description

architecture rtl of ss_u2p3 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
```

```
---------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

---------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
---------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
```

```
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state    : three_states := s30;
  signal next_rd_state       : three_states := s31;

-------------------------------------------------------------------------------
begin
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
  when 5 => next_rd_state <= s31;  -- ****************************
  when 7 => next_rd_state <= s32;  -- ****************************
  when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;

 RD_EN_DECODE:process(present_rd_state)
 begin
  case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
  end case;
 end process RD_EN_DECODE;

 RD_EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
        present_rd_state <= s30;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
  end if;
 end process RD_EN_STATE_REGISTER;

 RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   RD_CONT <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
```

```
   RD_CONT <= rd_enable;
  end if;
  end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------


  START2_PROCESS:process(MRST, MCLK)
  begin
   if ( MRST = '1') then
        start2 <= '1';
   elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        start2 <= START;
   end if;
  end process START2_PROCESS;

  INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
   begin
    if ( START = '0' ) then -- Initializing local phase counters
       fir_count <= 0;
         fft1_count <= 0;
       fft2_count <= 0;
         spec_count <= 0;
       fir_en2 <= '0';
       fft1_en2 <= '0';
       fft2_en2 <= '0';
       spec_en2 <= '0';
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
       fir_en2 <= fir_en;
       fft1_en2 <= fft1_en;
       fft2_en2 <= fft2_en;
       spec_en2 <= spec_en;
-- The followings are counter for EN*
      if (fir_en = '1') then
       fir_count <= fir_count + 1;
      end if;
      if (fft1_en = '1') then
       fft1_count <= fft1_count + 1;
      end if;
      if (fft2_en = '1') then
       fft2_count <= fft2_count + 1;
      end if;
      if (spec_en = '1') then
       spec_count <= spec_count + 1;
      end if;
    end if;
```

```
  end process INIT_SEQ;

 INIT_SEQ2:process(START2,MCLK)
 begin
 if ( START2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

 elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
 end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------------

 FFT11_EN_STATE_DECODE:process(fft1_count2)
 begin
 next_fft11_state2 <= ss10;
 case fft1_count2 is
  when 1 => next_fft11_state2 <= ss11;  -- ****************************
  when 3 => next_fft11_state2 <= ss11;  -- ****************************
  when 6 => next_fft11_state2 <= ss11;  -- ****************************
  when 8 => next_fft11_state2 <= ss11;  -- ****************************
  when OTHERS => next_fft11_state2 <= ss10;
 end case;
 end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------------
 FFT21_EN_STATE_DECODE:process(fft2_count2)
 begin
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 2 => next_fft21_state2 <= ss21;  -- ****************************
  when 7 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
```

```
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, START)
 begin
  if (START = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
       if ( fft1_en = '1' ) then
         present_fft11_state2 <= next_fft11_state2;
       end if;
       if ( fft2_en = '1' ) then
         present_fft21_state2 <= next_fft21_state2;
       end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
   when ss10 => ENABLE(0) <= '1'; -- Enable nothing
   when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
   when others => ENABLE(0) <= '1'; -- Enable nothing
  end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
```

```
  when 1 => next_fft21_state <= s11;  -- ****************************
  when 6 => next_fft21_state <= s11;  -- ****************************
  when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
        if ( FFT1_EN2 = '1' ) then
          present_fft21_state <= next_fft21_state;
        end if;
  end if;
 end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------------

 FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START)
 begin
 if ( START = '0' ) then
   FFT_SEL <= ( OTHERS => '0');
   INPUT_DATA_SEL <= ( OTHERS => '0');
   IQ_RAW_SEL <= '0';
   FIR_SEL <= '0';
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
   FFT_SEL <= mux_sel2;
   INPUT_DATA_SEL <= mux_sel;
   IQ_RAW_SEL <= mux_sel3;
   FIR_SEL <= OUTSEL;
 end if;
 end process OUTPUT_REGISTER_EN;
```

```
--------------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
 begin
 if ( START2 = '0' ) then
   OUT_EN <= ( OTHERS => '1');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   OUT_EN <= ENABLE;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;

--------------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
--------------------------------------------------------------------------------

 INPUT_DATA_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (spec_en = '1' ) then
    mux_sel <= "11";
   elsif ( fft2_en = '1') then
    mux_sel <= "10";
   elsif ( fft1_en = '1') then
    mux_sel <= "10";
   else
    case fir_count is
     when 1 => mux_sel <= "00"; --**********************************
     when 12 => mux_sel <= "00"; --**********************************
     when others => mux_sel <= "01";
    end case;
   end if;
  end if;
 end process INPUT_DATA_MUX;

--------------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
--------------------------------------------------------------------------------

 IQ_RAW_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel3 <= '0';
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fir_en = '1') then
    case fir_count is
```

```
      when 11 => mux_sel3 <= '1';
      when 22 => mux_sel3 <= '1';
      when others => mux_sel3 <= '0';
     end case;
    end if;
  end if;
 end process IQ_RAW_MUX;

-------------------------------------------------------------------------
-- FFT MUX PROCESSES ************* Each Sequencer has own procedure ********
-------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft2_en = '1') then
     if (fft2_count < 3) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 4) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 5) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 6) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 8) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 9) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 10) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 11) then --**********************************
      mux_sel2 <= "11";
     end if;
   elsif ( fft1_en = '1') then --**********************************
     if (fft1_count < 6) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 11) then --**********************************
      mux_sel2 <= "01";
     end if;
   end if;
  end if;
 end process FFT_MUX;

-------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------
```

```
FFT_EN_PROCESS:process(fft1_en, fft2_en)
begin
 fft_en <= fft1_en OR fft2_en;
end process FFT_EN_PROCESS;

FFT_WN_EN_PROCESS:process(START, MCLK)
begin
 if (START = '0') then
    fft_count_en <= '0';
 elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fft_en = '1') then
   if   (fft1_count2 < 2 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft1_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft1_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft1_count2 < 11 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft2_count2 < 2) then --*********************************
    fft_count_en <= '0';
   elsif (fft2_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft2_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft2_count2 < 11 ) then  --*********************************
    fft_count_en <= '1';
   else
    fft_count_en <= '0';
   end if;
  end if;
end if;
end process FFT_WN_EN_PROCESS;

FFT_INC_CLK_PROCESS:process(MCLK, START)
begin
if (START = '0') then
 fft_current_add <= (OTHERS => '0');
elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft_count_en = '1' ) then
      fft_current_add <= fft_current_add + "0001";
   end if;
end if;
end process FFT_INC_CLK_PROCESS;

FFT_WN_ADD_CLK:process(MCLK)
begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
```

```
        FFT_WN_ADD <= fft_current_add;
    end if;
  end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 ADD_STORE_EN_PROCESS:process(fir_count2)
 begin
  if (fir_count2 = 3) then
     add_store_en <= '1';
  else
     add_store_en <= '0';
  end if;
 end process ADD_STORE_EN_PROCESS;

 RD_RAM_LD_PROCESS:process(MRST, MCLK)
 begin
  if (MRST = '1') then
   ram_initial_add <= (others => '0');
  elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
   if ( add_store_en = '1') then
    ram_initial_add <= rd_current_add(3 downto 0);
   end if;
  end if;
 end process RD_RAM_LD_PROCESS;

 RD_RAM_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en2 = '1') then
    if (fir_count2 < 2 ) then
     ram_count_en <= '0';
     iq_state <= '0';
    elsif (fir_count2 < 11) then
     ram_count_en <= '1';
     iq_state <= '0';
    elsif (fir_count2 = 11) then
     ram_count_en <= '0';
     iq_state <= '0';
    elsif (fir_count2 < 22 ) then
     ram_count_en <= '1';
     iq_state <= '1';
    else
     ram_count_en <= '0';
     iq_state <= '0';
    end if;
   end if;
```

```
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (START2 = '0') then
     rd_current_add(3 downto 0) <= ram_initial_add;
     rd_current_add(4) <= '0';
   elsif (ram_count_en = '1' ) then
    if ( rd_current_add(3 downto 0) = "1001") then
        rd_current_add(3 downto 0) <= "0000";
        rd_current_add(4) <= iq_state;
    else
      rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
        rd_current_add(4) <= iq_state;
    end if;
   end if;
 end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  RD_RAM_ADD <= rd_current_add;
 end if;
end process RD_RAM_ADD_OUT_PROCESS;

-------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------

LOCAL_WN_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en = '1') then
   case fir_count is
    when 4 => local_count_en <= '1'; --*********************************
    when 15 => local_count_en <= '1'; --*********************************
    when others => local_count_en <= '0';
   end case;
  end if;
 end if;
end process LOCAL_WN_EN_PROCESS;

LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
begin
if ( PHASE_CLK = '0' ) then
```

```
      local_current_add <= ( OTHERS => '0');
    elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
      if (local_count_en = '1' ) then
       if ( local_current_add = "111" ) then
          local_current_add <= "000";
       else
        local_current_add <= local_current_add + "001";
       end if;
      end if;
    end if;
    end process LOCAL_INC_CLK_PROCESS;

    LOCAL_WN_ADD_CLK:process(MCLK)
    begin
     if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
          LOCAL_WN_ADD <= local_current_add;
     end if;
    end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------------
end rtl;
```

第4項　ss_u3 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.
```

```
library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u3p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u3p3;

-- ss_u3p3 Architecture Description

architecture rtl of ss_u3p3 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
```

```
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;

-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
```

```
RD_EN_STATE_DECODE:process(fir_count2)
begin
next_rd_state <= s30;
case fir_count2 is
 when 9 => next_rd_state <= s31;  -- ****************************
 when 11 => next_rd_state <= s32;  -- ****************************
 when OTHERS => next_rd_state <= s30;
end case;
end process RD_EN_STATE_DECODE;

RD_EN_DECODE:process(present_rd_state)
begin
 case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
 end case;
end process RD_EN_DECODE;

RD_EN_STATE_REGISTER:process(MCLK, start)
begin
 if (start = '0') then
        present_rd_state <= s30;
 elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
 end if;
end process RD_EN_STATE_REGISTER;

RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
begin
if ( start2 = '0' ) then
  RD_CONT <= ( OTHERS => '0');
elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
  RD_CONT <= rd_enable;
end if;
end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
       start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
```

```
    end if;
  end process START2_PROCESS;

  INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
   begin
    if ( START = '0' ) then -- Initializing local phase counters
      fir_count <= 0;
        fft1_count <= 0;
      fft2_count <= 0;
        spec_count <= 0;
      fir_en2 <= '0';
      fft1_en2 <= '0';
      fft2_en2 <= '0';
      spec_en2 <= '0';
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
      fir_en2 <= fir_en;
      fft1_en2 <= fft1_en;
      fft2_en2 <= fft2_en;
      spec_en2 <= spec_en;
-- The followings are counter for EN*
      if (fir_en = '1') then
       fir_count <= fir_count + 1;
      end if;
      if (fft1_en = '1') then
       fft1_count <= fft1_count + 1;
      end if;
      if (fft2_en = '1') then
       fft2_count <= fft2_count + 1;
      end if;
      if (spec_en = '1') then
       spec_count <= spec_count + 1;
      end if;
    end if;
  end process INIT_SEQ;

  INIT_SEQ2:process(START2,MCLK)
  begin
  if ( START2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

  elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
```

```
 end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------

 FFT11_EN_STATE_DECODE:process(fft1_count2)
 begin
 next_fft11_state2 <= ss10;
 case fft1_count2 is
  when 1 => next_fft11_state2 <= ss11;  -- ****************************
  when 3 => next_fft11_state2 <= ss11;  -- ****************************
  when 6 => next_fft11_state2 <= ss11;  -- ****************************
  when 8 => next_fft11_state2 <= ss11;  -- ****************************
  when OTHERS => next_fft11_state2 <= ss10;
 end case;
 end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------
 FFT21_EN_STATE_DECODE:process(fft2_count2)
 begin
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 3 => next_fft21_state2 <= ss21;  -- ****************************
  when 8 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, START)
 begin
  if (START = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
       if ( fft1_en = '1' ) then
         present_fft11_state2 <= next_fft11_state2;
       end if;
       if ( fft2_en = '1' ) then
         present_fft21_state2 <= next_fft21_state2;
       end if;
```

```
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
   when ss10 => ENABLE(0) <= '1'; -- Enable nothing
   when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
   when others => ENABLE(0) <= '1'; -- Enable nothing
  end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
   when 1 => next_fft21_state <= s11;  -- ****************************
   when 6 => next_fft21_state <= s11;  -- ****************************
   when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
       if ( FFT1_EN2 = '1' ) then
```

```
          present_fft21_state <= next_fft21_state;
        end if;
    end if;
  end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

  FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
  begin
  case present_fft21_state is
    when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
    when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
  end case;
  end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

  OUTPUT_REGISTER_EN:process(MCLK, START)
  begin
  if ( START = '0' ) then
    FFT_SEL <= ( OTHERS => '0');
    INPUT_DATA_SEL <= ( OTHERS => '0');
    IQ_RAW_SEL <= '0';
    FIR_SEL <= '0';
  elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
    FFT_SEL <= mux_sel2;
    INPUT_DATA_SEL <= mux_sel;
    IQ_RAW_SEL <= mux_sel3;
    FIR_SEL <= OUTSEL;
  end if;
  end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------

  OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
  begin
  if ( START2 = '0' ) then
    OUT_EN <= ( OTHERS => '1');
  elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
    OUT_EN <= ENABLE;
  end if;
  end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------
```

```
INPUT_DATA_MUX:process(START2, MCLK)
begin
 if (START2= '0') then
  mux_sel <= ( others => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (spec_en = '1' ) then
   mux_sel <= "11";
  elsif ( fft2_en = '1') then
   mux_sel <= "10";
  elsif ( fft1_en = '1') then
   mux_sel <= "10";
  else
   case fir_count is
    when 1 => mux_sel <= "00"; --**********************************
    when 12 => mux_sel <= "00"; --**********************************
    when others => mux_sel <= "01";
   end case;
  end if;
 end if;
end process INPUT_DATA_MUX;

-------------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------------

IQ_RAW_MUX:process(START2, MCLK)
begin
 if (START2= '0') then
  mux_sel3 <= '0';
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (fir_en = '1') then
   case fir_count is
    when 11 => mux_sel3 <= '1';
    when 22 => mux_sel3 <= '1';
    when others => mux_sel3 <= '0';
   end case;
  end if;
 end if;
end process IQ_RAW_MUX;

-------------------------------------------------------------------------------
-- FFT MUX PROCESSES ********** Each Sequencer has own procedure ********
-------------------------------------------------------------------------------

FFT_MUX:process(START2, MCLK)
begin
 if (START2 = '0') then
```

```
    mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fft2_en = '1') then
     if (fft2_count < 6) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 11) then --**********************************
      mux_sel2 <= "11";
     end if;
    elsif ( fft1_en = '1') then --**********************************
     if (fft1_count < 6) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 11) then --**********************************
      mux_sel2 <= "01";
     end if;
    end if;
  end if;
 end process FFT_MUX;

-------------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
     fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fft_en = '1') then
    if    (fft1_count2 < 2 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft1_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 11 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 2) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 11 ) then  --**********************************
     fft_count_en <= '1';
```

```
    else
      fft_count_en <= '0';
    end if;
   end if;
  end if;
  end process FFT_WN_EN_PROCESS;

  FFT_INC_CLK_PROCESS:process(MCLK, START)
  begin
  if (START = '0') then
   fft_current_add <= (OTHERS => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
     if (fft_count_en = '1' ) then
        fft_current_add <= fft_current_add + "0001";
     end if;
  end if;
  end process FFT_INC_CLK_PROCESS;

  FFT_WN_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
         FFT_WN_ADD <= fft_current_add;
    end if;
  end process FFT_WN_ADD_CLK;

 ---------------------------------------------------------------------------
 -- Ich, Qch RAM CONTROL PROCESSES
 ---------------------------------------------------------------------------
  ADD_STORE_EN_PROCESS:process(fir_count2)
  begin
  if (fir_count2 = 3) then
     add_store_en <= '1';
  else
     add_store_en <= '0';
  end if;
 end process ADD_STORE_EN_PROCESS;

  RD_RAM_LD_PROCESS:process(MRST, MCLK)
  begin
  if (MRST = '1') then
   ram_initial_add <= (others => '0');
  elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
   if ( add_store_en = '1') then
    ram_initial_add <= rd_current_add(3 downto 0);
   end if;
  end if;
 end process RD_RAM_LD_PROCESS;
```

```
 RD_RAM_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en2 = '1') then
    if (fir_count2 < 2 ) then
     ram_count_en <= '0';
     iq_state <= '0';
    elsif (fir_count2 < 11) then
     ram_count_en <= '1';
     iq_state <= '0';
    elsif (fir_count2 = 11) then
     ram_count_en <= '0';
     iq_state <= '0';
    elsif (fir_count2 < 22 ) then
     ram_count_en <= '1';
     iq_state <= '1';
    else
     ram_count_en <= '0';
     iq_state <= '0';
    end if;
   end if;
  end if;
 end process RD_RAM_EN_PROCESS;

 RD_RAM_INC_CLK_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (START2 = '0') then
     rd_current_add(3 downto 0) <= ram_initial_add;
     rd_current_add(4) <= '0';
    elsif (ram_count_en = '1' ) then
     if ( rd_current_add(3 downto 0) = "1001") then
         rd_current_add(3 downto 0) <= "0000";
         rd_current_add(4) <= iq_state;
     else
       rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
         rd_current_add(4) <= iq_state;
     end if;
    end if;
  end if;
 end process RD_RAM_INC_CLK_PROCESS;

 RD_RAM_ADD_OUT_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   RD_RAM_ADD <= rd_current_add;
  end if;
 end process RD_RAM_ADD_OUT_PROCESS;
```

```
-------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------

 LOCAL_WN_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en = '1') then
    case fir_count is
     when 4 => local_count_en <= '1'; --*********************************
     when 15 => local_count_en <= '1'; --*********************************
     when others => local_count_en <= '0';
    end case;
   end if;
  end if;
 end process LOCAL_WN_EN_PROCESS;

 LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
 begin
 if ( PHASE_CLK = '0' ) then
    local_current_add <= ( OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (local_count_en = '1' ) then
     if ( local_current_add = "111" ) then
        local_current_add <= "000";
     else
      local_current_add <= local_current_add + "001";
     end if;
    end if;
 end if;
 end process LOCAL_INC_CLK_PROCESS;

 LOCAL_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       LOCAL_WN_ADD <= local_current_add;
   end if;
 end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------
end rtl;
```

第5項　ss_u4 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
```

```
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u4p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u4p3;

-- ss_u4p3 Architecture Description

architecture rtl of ss_u4p3 is

------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
```

```
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
```

```
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;

-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
  when 13 => next_rd_state <= s31;  -- ****************************
  when 15 => next_rd_state <= s32;  -- ****************************
  when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;

 RD_EN_DECODE:process(present_rd_state)
 begin
  case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
  end case;
 end process RD_EN_DECODE;

 RD_EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
        present_rd_state <= s30;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
  end if;
```

```
 end process RD_EN_STATE_REGISTER;

 RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   RD_CONT <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   RD_CONT <= rd_enable;
 end if;
 end process RD_OUTPUT_REGISTER_TRIGGER;

--------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
--------------------------------------------------------------------------
--------------------------------------------------------------------------
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
       start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
  end if;
 end process START2_PROCESS;

 INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
  begin
   if ( START = '0' ) then -- Initializing local phase counters
      fir_count <= 0;
        fft1_count <= 0;
      fft2_count <= 0;
        spec_count <= 0;
      fir_en2 <= '0';
      fft1_en2 <= '0';
      fft2_en2 <= '0';
      spec_en2 <= '0';
   elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
      fir_en2 <= fir_en;
      fft1_en2 <= fft1_en;
      fft2_en2 <= fft2_en;
      spec_en2 <= spec_en;
-- The followings are counter for EN*
      if (fir_en = '1') then
       fir_count <= fir_count + 1;
      end if;
      if (fft1_en = '1') then
       fft1_count <= fft1_count + 1;
      end if;
      if (fft2_en = '1') then
```

```
        fft2_count <= fft2_count + 1;
       end if;
       if (spec_en = '1') then
        spec_count <= spec_count + 1;
       end if;
     end if;
  end process INIT_SEQ;

 INIT_SEQ2:process(START2,MCLK)
 begin
 if ( START2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

 elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
 end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------

 FFT11_EN_STATE_DECODE:process(fft1_count2)
 begin
 next_fft11_state2 <= ss10;
 case fft1_count2 is
  when 1 => next_fft11_state2 <= ss11;  -- ****************************
  when 3 => next_fft11_state2 <= ss11;  -- ****************************
  when 6 => next_fft11_state2 <= ss11;  -- ****************************
  when 8 => next_fft11_state2 <= ss11;  -- ****************************
  when OTHERS => next_fft11_state2 <= ss10;
 end case;
 end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------
 FFT21_EN_STATE_DECODE:process(fft2_count2)
 begin
```

```
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 4 => next_fft21_state2 <= ss21;  -- ****************************
  when 9 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, START)
 begin
  if (START = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
       if ( fft1_en = '1' ) then
         present_fft11_state2 <= next_fft11_state2;
       end if;
       if ( fft2_en = '1' ) then
         present_fft21_state2 <= next_fft21_state2;
       end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
  when ss10 => ENABLE(0) <= '1'; -- Enable nothing
  when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
  when others => ENABLE(0) <= '1'; -- Enable nothing
 end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
  when ss20 => ENABLE(1) <= '1'; -- Enable nothing
  when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
  when others => ENABLE(1) <= '1'; -- Enable nothing
 end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
```

```
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
  when 1 => next_fft21_state <= s11;  -- ***************************
  when 6 => next_fft21_state <= s11;  -- ***************************
  when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
       if ( FFT1_EN2 = '1' ) then
         present_fft21_state <= next_fft21_state;
       end if;
  end if;
 end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START)
 begin
 if ( START = '0' ) then
   FFT_SEL <= ( OTHERS => '0');
   INPUT_DATA_SEL <= ( OTHERS => '0');
   IQ_RAW_SEL <= '0';
   FIR_SEL <= '0';
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
```

```
    FFT_SEL <= mux_sel2;
    INPUT_DATA_SEL <= mux_sel;
    IQ_RAW_SEL <= mux_sel3;
    FIR_SEL <= OUTSEL;
  end if;
  end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------------

  OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
  begin
  if ( START2 = '0' ) then
    OUT_EN <= ( OTHERS => '1');
  elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
    OUT_EN <= ENABLE;
  end if;
  end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------------

  INPUT_DATA_MUX:process(START2, MCLK)
  begin
   if (START2= '0') then
    mux_sel <= ( others => '0');
   elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (spec_en = '1' ) then
     mux_sel <= "11";
    elsif ( fft2_en = '1') then
     mux_sel <= "10";
    elsif ( fft1_en = '1') then
     mux_sel <= "10";
    else
     case fir_count is
      when 1 => mux_sel <= "00"; --**********************************
      when 12 => mux_sel <= "00"; --**********************************
      when others => mux_sel <= "01";
     end case;
    end if;
   end if;
  end process INPUT_DATA_MUX;

-------------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------------

  IQ_RAW_MUX:process(START2, MCLK)
```

```
 begin
  if (START2= '0') then
    mux_sel3 <= '0';
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fir_en = '1') then
    case fir_count is
     when 11 => mux_sel3 <= '1';
     when 22 => mux_sel3 <= '1';
     when others => mux_sel3 <= '0';
    end case;
   end if;
  end if;
 end process IQ_RAW_MUX;

 ---------------------------------------------------------------------------
 -- FFT MUX PROCESSES ************* Each Sequencer has own procedure ********
 ---------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
    mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fft2_en = '1') then
     if (fft2_count < 3) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 4) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 5) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 6) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 8) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 9) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 10) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 11) then --**********************************
      mux_sel2 <= "11";
     end if;
    elsif ( fft1_en = '1') then --**********************************
     if (fft1_count < 6) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 11) then --**********************************
      mux_sel2 <= "01";
     end if;
    end if;
```

```
 end if;
end process FFT_MUX;

-------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------
FFT_EN_PROCESS:process(fft1_en, fft2_en)
begin
  fft_en <= fft1_en OR fft2_en;
end process FFT_EN_PROCESS;

FFT_WN_EN_PROCESS:process(START, MCLK)
begin
 if (START = '0') then
    fft_count_en <= '0';
 elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fft_en = '1') then
   if   (fft1_count2 < 2 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft1_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft1_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft1_count2 < 11 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft2_count2 < 2) then --*********************************
    fft_count_en <= '0';
   elsif (fft2_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
   elsif (fft2_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
   elsif (fft2_count2 < 11 ) then  --*********************************
    fft_count_en <= '1';
   else
    fft_count_en <= '0';
   end if;
  end if;
end if;
end process FFT_WN_EN_PROCESS;

FFT_INC_CLK_PROCESS:process(MCLK, START)
begin
if (START = '0') then
  fft_current_add <= (OTHERS => '0');
elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft_count_en = '1' ) then
      fft_current_add <= fft_current_add + "0001";
   end if;
```

```
 end if;
 end process FFT_INC_CLK_PROCESS;

 FFT_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        FFT_WN_ADD <= fft_current_add;
   end if;
 end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------

 ADD_STORE_EN_PROCESS:process(fir_count2)
 begin
  if (fir_count2 = 3) then
     add_store_en <= '1';
  else
     add_store_en <= '0';
  end if;
 end process ADD_STORE_EN_PROCESS;

 RD_RAM_LD_PROCESS:process(MRST, MCLK)
 begin
  if (MRST = '1') then
    ram_initial_add <= (others => '0');
  elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
    if ( add_store_en = '1') then
     ram_initial_add <= rd_current_add(3 downto 0);
    end if;
  end if;
 end process RD_RAM_LD_PROCESS;

 RD_RAM_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
    if (fir_en2 = '1') then
      if (fir_count2 < 2 ) then
        ram_count_en <= '0';
        iq_state <= '0';
      elsif (fir_count2 < 11) then
        ram_count_en <= '1';
        iq_state <= '0';
      elsif (fir_count2 = 11) then
        ram_count_en <= '0';
        iq_state <= '0';
      elsif (fir_count2 < 22 ) then
```

```
      ram_count_en <= '1';
      iq_state <= '1';
     else
      ram_count_en <= '0';
      iq_state <= '0';
     end if;
   end if;
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (START2 = '0') then
     rd_current_add(3 downto 0) <= ram_initial_add;
     rd_current_add(4) <= '0';
   elsif (ram_count_en = '1' ) then
    if ( rd_current_add(3 downto 0) = "1001") then
        rd_current_add(3 downto 0) <= "0000";
        rd_current_add(4) <= iq_state;
    else
      rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
        rd_current_add(4) <= iq_state;
    end if;
   end if;
 end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  RD_RAM_ADD <= rd_current_add;
 end if;
end process RD_RAM_ADD_OUT_PROCESS;

-------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------

LOCAL_WN_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en = '1') then
   case fir_count is
    when 4 => local_count_en <= '1'; --*********************************
    when 15 => local_count_en <= '1'; --*********************************
    when others => local_count_en <= '0';
   end case;
```

```
  end if;
 end if;
end process LOCAL_WN_EN_PROCESS;

 LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
 begin
 if ( PHASE_CLK = '0' ) then
   local_current_add <= ( OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (local_count_en = '1' ) then
    if ( local_current_add = "111" ) then
       local_current_add <= "000";
    else
     local_current_add <= local_current_add + "001";
    end if;
   end if;
 end if;
 end process LOCAL_INC_CLK_PROCESS;

 LOCAL_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       LOCAL_WN_ADD <= local_current_add;
   end if;
 end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------
end rtl;
```

第6項　ss_u5 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
```

```
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u5p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u5p3;

-- ss_u5p3 Architecture Description

architecture rtl of ss_u5p3 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
```

```
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;
```

```
-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
  when 2 => next_rd_state <= s31;  -- ****************************
  when 4 => next_rd_state <= s32;  -- ****************************
  when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;

 RD_EN_DECODE:process(present_rd_state)
 begin
  case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
  end case;
 end process RD_EN_DECODE;

 RD_EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
        present_rd_state <= s30;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
  end if;
 end process RD_EN_STATE_REGISTER;

 RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   RD_CONT <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   RD_CONT <= rd_enable;
 end if;
 end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
       start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
  end if;
 end process START2_PROCESS;

 INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
  begin
   if ( START = '0' ) then -- Initializing local phase counters
     fir_count <= 0;
       fft1_count <= 0;
     fft2_count <= 0;
       spec_count <= 0;
     fir_en2 <= '0';
     fft1_en2 <= '0';
     fft2_en2 <= '0';
     spec_en2 <= '0';
   elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
     fir_en2 <= fir_en;
     fft1_en2 <= fft1_en;
     fft2_en2 <= fft2_en;
     spec_en2 <= spec_en;
-- The followings are counter for EN*
     if (fir_en = '1') then
      fir_count <= fir_count + 1;
     end if;
     if (fft1_en = '1') then
      fft1_count <= fft1_count + 1;
     end if;
     if (fft2_en = '1') then
      fft2_count <= fft2_count + 1;
     end if;
     if (spec_en = '1') then
      spec_count <= spec_count + 1;
     end if;
   end if;
 end process INIT_SEQ;

INIT_SEQ2:process(START2,MCLK)
begin
if ( START2 = '0' ) then
  fir_count2 <= 0;
  fft1_count2 <= 0;
  fft2_count2 <= 0;
```

```
    spec_count2 <= 0;

  elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
  end if;
  end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------

  FFT11_EN_STATE_DECODE:process(fft1_count2)
  begin
  next_fft11_state2 <= ss10;
  case fft1_count2 is
    when 2 => next_fft11_state2 <= ss11;  -- ****************************
    when 4 => next_fft11_state2 <= ss11;  -- ****************************
    when 7 => next_fft11_state2 <= ss11;  -- ****************************
    when 9 => next_fft11_state2 <= ss11;  -- ****************************
    when OTHERS => next_fft11_state2 <= ss10;
  end case;
  end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------
  FFT21_EN_STATE_DECODE:process(fft2_count2)
  begin
  next_fft21_state2 <= ss20;
  case fft2_count2 is
    when 1 => next_fft21_state2 <= ss21;  -- ****************************
    when 6 => next_fft21_state2 <= ss21;  -- ****************************
    when OTHERS => next_fft21_state2 <= ss20;
  end case;
  end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
  EN_STATE_REGISTER:process(MCLK, START)
  begin
    if (START = '0') then
        present_fft11_state2 <= ss10;
        present_fft21_state2 <= ss20;
```

```
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        if ( fft1_en = '1' ) then
          present_fft11_state2 <= next_fft11_state2;
        end if;
        if ( fft2_en = '1' ) then
          present_fft21_state2 <= next_fft21_state2;
        end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
   when ss10 => ENABLE(0) <= '1'; -- Enable nothing
   when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
   when others => ENABLE(0) <= '1'; -- Enable nothing
  end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
   when 2 => next_fft21_state <= s11;  -- ***************************
   when 7 => next_fft21_state <= s11;  -- ***************************
   when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------
```

```
TRIG_STATE_REGISTER:process(MCLK, START2)

begin
 if (START2 = '0') then
     present_fft21_state <= s10;
 elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
       if ( FFT1_EN2 = '1' ) then
         present_fft21_state <= next_fft21_state;
       end if;
 end if;
end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
begin
case present_fft21_state is
 when s10 => OUTSEL <= '0'; -- SEL FIR CH.B
 when s11 => OUTSEL <= '1'; -- SEL FIR CH.A
end case;
end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

OUTPUT_REGISTER_EN:process(MCLK, START)
begin
if ( START = '0' ) then
  FFT_SEL <= ( OTHERS => '0');
  INPUT_DATA_SEL <= ( OTHERS => '0');
  IQ_RAW_SEL <= '0';
  FIR_SEL <= '0';
elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
  FFT_SEL <= mux_sel2;
  INPUT_DATA_SEL <= mux_sel;
  IQ_RAW_SEL <= mux_sel3;
  FIR_SEL <= OUTSEL;
end if;
end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------

OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
begin
if ( START2 = '0' ) then
  OUT_EN <= ( OTHERS => '1');
elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
```

```
   OUT_EN <= ENABLE;
  end if;
  end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------

  INPUT_DATA_MUX:process(START2, MCLK)
  begin
   if (START2= '0') then
    mux_sel <= ( others => '0');
   elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (spec_en = '1' ) then
     mux_sel <= "11";
    elsif ( fft2_en = '1') then
     mux_sel <= "10";
    elsif ( fft1_en = '1') then
     mux_sel <= "10";
    else
     case fir_count is
      when 1 => mux_sel <= "00"; --**********************************
      when 12 => mux_sel <= "00"; --**********************************
      when others => mux_sel <= "01";
     end case;
    end if;
   end if;
  end process INPUT_DATA_MUX;

-------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------

  IQ_RAW_MUX:process(START2, MCLK)
  begin
   if (START2= '0') then
    mux_sel3 <= '0';
   elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fir_en = '1') then
     case fir_count is
      when 11 => mux_sel3 <= '1';
      when 22 => mux_sel3 <= '1';
      when others => mux_sel3 <= '0';
     end case;
    end if;
   end if;
  end process IQ_RAW_MUX;
```

```
-------------------------------------------------------------------------------
-- FFT MUX PROCESSES ************* Each Sequencer has own procedure ********
-------------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft2_en = '1') then
    if (fft2_count < 6) then --**********************************
     mux_sel2 <= "10";
    elsif (fft2_count < 11) then --**********************************
     mux_sel2 <= "11";
    end if;
   elsif ( fft1_en = '1') then --**********************************
    if (fft1_count < 3) then --**********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 4) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 5) then --**********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 6) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 8) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 9) then --**********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 10) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 11) then --**********************************
     mux_sel2 <= "01";
    end if;
   end if;
  end if;
 end process FFT_MUX;

-------------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
```

```
    fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fft_en = '1') then
    if   (fft1_count2 < 2 ) then --*********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 6 ) then --*********************************
     fft_count_en <= '1';
    elsif (fft1_count2 < 7 ) then --*********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 11 ) then --*********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 2) then --*********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 6 ) then --*********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 7 ) then --*********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 11 ) then  --*********************************
     fft_count_en <= '1';
    else
     fft_count_en <= '0';
    end if;
   end if;
  end if;
  end process FFT_WN_EN_PROCESS;

  FFT_INC_CLK_PROCESS:process(MCLK, START)
  begin
  if (START = '0') then
   fft_current_add <= (OTHERS => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
     if (fft_count_en = '1' ) then
        fft_current_add <= fft_current_add + "0001";
     end if;
  end if;
  end process FFT_INC_CLK_PROCESS;

  FFT_WN_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
          FFT_WN_ADD <= fft_current_add;
    end if;
  end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 ADD_STORE_EN_PROCESS:process(fir_count2)
```

```
begin
 if (fir_count2 = 3) then
    add_store_en <= '1';
 else
    add_store_en <= '0';
 end if;
end process ADD_STORE_EN_PROCESS;

RD_RAM_LD_PROCESS:process(MRST, MCLK)
begin
 if (MRST = '1') then
  ram_initial_add <= (others => '0');
 elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
  if ( add_store_en = '1') then
   ram_initial_add <= rd_current_add(3 downto 0);
  end if;
 end if;
end process RD_RAM_LD_PROCESS;

RD_RAM_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en2 = '1') then
   if (fir_count2 < 2 ) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 11) then
    ram_count_en <= '1';
    iq_state <= '0';
   elsif (fir_count2 = 11) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 22 ) then
    ram_count_en <= '1';
    iq_state <= '1';
   else
    ram_count_en <= '0';
    iq_state <= '0';
   end if;
  end if;
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (START2 = '0') then
   rd_current_add(3 downto 0) <= ram_initial_add;
```

```
      rd_current_add(4) <= '0';
    elsif (ram_count_en = '1' ) then
     if ( rd_current_add(3 downto 0) = "1001") then
         rd_current_add(3 downto 0) <= "0000";
         rd_current_add(4) <= iq_state;
     else
       rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
         rd_current_add(4) <= iq_state;
     end if;
    end if;
 end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  RD_RAM_ADD <= rd_current_add;
 end if;
end process RD_RAM_ADD_OUT_PROCESS;

--------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
--------------------------------------------------------------------------

LOCAL_WN_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en = '1') then
   case fir_count is
    when 4 => local_count_en <= '1'; --*********************************
    when 15 => local_count_en <= '1'; --*********************************
    when others => local_count_en <= '0';
   end case;
  end if;
 end if;
end process LOCAL_WN_EN_PROCESS;

LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
begin
if ( PHASE_CLK = '0' ) then
   local_current_add <= ( OTHERS => '0');
elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (local_count_en = '1' ) then
    if ( local_current_add = "111" ) then
       local_current_add <= "000";
    else
     local_current_add <= local_current_add + "001";
    end if;
```

```
    end if;
  end if;
  end process LOCAL_INC_CLK_PROCESS;

  LOCAL_WN_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        LOCAL_WN_ADD <= local_current_add;
    end if;
  end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------
end rtl;
```

第7項　ss_u6 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u6p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
```

```
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u6p3;

-- ss_u6p3 Architecture Description

architecture rtl of ss_u6p3 is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
```

```
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;

-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
   when 6 => next_rd_state <= s31;  -- ****************************
   when 8 => next_rd_state <= s32;  -- ****************************
   when OTHERS => next_rd_state <= s30;
```

```
end case;
end process RD_EN_STATE_DECODE;

RD_EN_DECODE:process(present_rd_state)
begin
 case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
 end case;
end process RD_EN_DECODE;

RD_EN_STATE_REGISTER:process(MCLK, start)
begin
 if (start = '0') then
        present_rd_state <= s30;
 elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
 end if;
end process RD_EN_STATE_REGISTER;

RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
begin
if ( start2 = '0' ) then
  RD_CONT <= ( OTHERS => '0');
elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
  RD_CONT <= rd_enable;
end if;
end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
       start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
  end if;
 end process START2_PROCESS;

 INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
  begin
   if ( START = '0' ) then -- Initializing local phase counters
     fir_count <= 0;
       fft1_count <= 0;
```

```
        fft2_count <= 0;
          spec_count <= 0;
        fir_en2 <= '0';
        fft1_en2 <= '0';
        fft2_en2 <= '0';
        spec_en2 <= '0';
      elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
        fir_en2 <= fir_en;
        fft1_en2 <= fft1_en;
        fft2_en2 <= fft2_en;
        spec_en2 <= spec_en;
-- The followings are counter for EN*
       if (fir_en = '1') then
        fir_count <= fir_count + 1;
       end if;
       if (fft1_en = '1') then
        fft1_count <= fft1_count + 1;
       end if;
       if (fft2_en = '1') then
        fft2_count <= fft2_count + 1;
       end if;
       if (spec_en = '1') then
        spec_count <= spec_count + 1;
       end if;
      end if;
   end process INIT_SEQ;

 INIT_SEQ2:process(START2,MCLK)
 begin
 if ( START2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

 elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
 end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL ****************** Each Sequencer has own procedure ********
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
```

```
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------

 FFT11_EN_STATE_DECODE:process(fft1_count2)
 begin
 next_fft11_state2 <= ss10;
 case fft1_count2 is
  when 2 => next_fft11_state2 <= ss11;  -- ****************************
  when 4 => next_fft11_state2 <= ss11;  -- ****************************
  when 7 => next_fft11_state2 <= ss11;  -- ****************************
  when 9 => next_fft11_state2 <= ss11;  -- ****************************
  when OTHERS => next_fft11_state2 <= ss10;
 end case;
 end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------
 FFT21_EN_STATE_DECODE:process(fft2_count2)
 begin
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 2 => next_fft21_state2 <= ss21;  -- ****************************
  when 7 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, START)
 begin
  if (START = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        if ( fft1_en = '1' ) then
          present_fft11_state2 <= next_fft11_state2;
        end if;
        if ( fft2_en = '1' ) then
          present_fft21_state2 <= next_fft21_state2;
        end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
```

```
  when ss10 => ENABLE(0) <= '1'; -- Enable nothing
  when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
  when others => ENABLE(0) <= '1'; -- Enable nothing
 end case;
end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
begin
 case present_fft21_state2 is
  when ss20 => ENABLE(1) <= '1'; -- Enable nothing
  when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
  when others => ENABLE(1) <= '1'; -- Enable nothing
 end case;
end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

FFT21_SEL_STATE_DECODE:process(fft1_count)
begin
next_fft21_state <= s10;
case fft1_count is
  when 2 => next_fft21_state <= s11;  -- ***************************
  when 7 => next_fft21_state <= s11;  -- ***************************
  when OTHERS => next_fft21_state <= s10;
end case;
end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------

TRIG_STATE_REGISTER:process(MCLK, START2)

begin
 if (START2 = '0') then
     present_fft21_state <= s10;
 elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
      if ( FFT1_EN2 = '1' ) then
        present_fft21_state <= next_fft21_state;
      end if;
 end if;
end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
```

```
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START)
 begin
 if ( START = '0' ) then
  FFT_SEL <= ( OTHERS => '0');
  INPUT_DATA_SEL <= ( OTHERS => '0');
  IQ_RAW_SEL <= '0';
  FIR_SEL <= '0';
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
  FFT_SEL <= mux_sel2;
  INPUT_DATA_SEL <= mux_sel;
  IQ_RAW_SEL <= mux_sel3;
  FIR_SEL <= OUTSEL;
 end if;
 end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
 begin
 if ( START2 = '0' ) then
  OUT_EN <= ( OTHERS => '1');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
  OUT_EN <= ENABLE;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------

 INPUT_DATA_MUX:process(START2, MCLK)
 begin
 if (START2= '0') then
  mux_sel <= ( others => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (spec_en = '1' ) then
   mux_sel <= "11";
```

```
  elsif ( fft2_en = '1') then
   mux_sel <= "10";
  elsif ( fft1_en = '1') then
   mux_sel <= "10";
  else
   case fir_count is
    when 1 => mux_sel <= "00"; --**********************************
    when 12 => mux_sel <= "00"; --**********************************
    when others => mux_sel <= "01";
   end case;
  end if;
 end if;
end process INPUT_DATA_MUX;

-------------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------------

IQ_RAW_MUX:process(START2, MCLK)
begin
 if (START2= '0') then
  mux_sel3 <= '0';
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (fir_en = '1') then
   case fir_count is
    when 11 => mux_sel3 <= '1';
    when 22 => mux_sel3 <= '1';
    when others => mux_sel3 <= '0';
   end case;
  end if;
 end if;
end process IQ_RAW_MUX;

-------------------------------------------------------------------------------
-- FFT MUX PROCESSES ********** Each Sequencer has own procedure ********
-------------------------------------------------------------------------------

FFT_MUX:process(START2, MCLK)
begin
 if (START2= '0') then
  mux_sel2 <= ( others => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft2_en = '1') then
    if (fft2_count < 3) then --**********************************
     mux_sel2 <= "11";
    elsif (fft2_count < 4) then --**********************************
     mux_sel2 <= "10";
    elsif (fft2_count < 5) then --**********************************
```

```
      mux_sel2 <= "11";
     elsif (fft2_count < 6) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 8) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 9) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 10) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 11) then --**********************************
      mux_sel2 <= "11";
     end if;
    elsif ( fft1_en = '1') then --**********************************
     if (fft1_count < 3) then --**********************************
      mux_sel2 <= "01";
     elsif (fft1_count < 4) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 5) then --**********************************
      mux_sel2 <= "01";
     elsif (fft1_count < 6) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 8) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 9) then --**********************************
      mux_sel2 <= "01";
     elsif (fft1_count < 10) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 11) then --**********************************
      mux_sel2 <= "01";
     end if;
    end if;
  end if;
 end process FFT_MUX;

-------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
     fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fft_en = '1') then
```

```
  if   (fft1_count2 < 2 ) then --*********************************
    fft_count_en <= '0';
  elsif (fft1_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
  elsif (fft1_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
  elsif (fft1_count2 < 11 ) then --*********************************
    fft_count_en <= '1';
  elsif (fft2_count2 < 2) then --*********************************
    fft_count_en <= '0';
  elsif (fft2_count2 < 6 ) then --*********************************
    fft_count_en <= '1';
  elsif (fft2_count2 < 7 ) then --*********************************
    fft_count_en <= '0';
  elsif (fft2_count2 < 11 ) then  --*********************************
    fft_count_en <= '1';
  else
    fft_count_en <= '0';
  end if;
 end if;
end if;
end process FFT_WN_EN_PROCESS;

FFT_INC_CLK_PROCESS:process(MCLK, START)
begin
if (START = '0') then
  fft_current_add <= (OTHERS => '0');
elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft_count_en = '1' ) then
      fft_current_add <= fft_current_add + "0001";
   end if;
end if;
end process FFT_INC_CLK_PROCESS;

FFT_WN_ADD_CLK:process(MCLK)
begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       FFT_WN_ADD <= fft_current_add;
  end if;
end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 ADD_STORE_EN_PROCESS:process(fir_count2)
 begin
 if (fir_count2 = 3) then
    add_store_en <= '1';
```

```
 else
    add_store_en <= '0';
 end if;
end process ADD_STORE_EN_PROCESS;

RD_RAM_LD_PROCESS:process(MRST, MCLK)
begin
 if (MRST = '1') then
  ram_initial_add <= (others => '0');
 elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
  if ( add_store_en = '1') then
   ram_initial_add <= rd_current_add(3 downto 0);
  end if;
 end if;
end process RD_RAM_LD_PROCESS;

RD_RAM_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en2 = '1') then
   if (fir_count2 < 2 ) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 11) then
    ram_count_en <= '1';
    iq_state <= '0';
   elsif (fir_count2 = 11) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 22 ) then
    ram_count_en <= '1';
    iq_state <= '1';
   else
    ram_count_en <= '0';
    iq_state <= '0';
   end if;
  end if;
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (START2 = '0') then
   rd_current_add(3 downto 0) <= ram_initial_add;
   rd_current_add(4) <= '0';
  elsif (ram_count_en = '1' ) then
   if ( rd_current_add(3 downto 0) = "1001") then
```

```
          rd_current_add(3 downto 0) <= "0000";
          rd_current_add(4) <= iq_state;
      else
        rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
          rd_current_add(4) <= iq_state;
      end if;
    end if;
  end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   RD_RAM_ADD <= rd_current_add;
 end if;
end process RD_RAM_ADD_OUT_PROCESS;

-------------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------------

 LOCAL_WN_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en = '1') then
     case fir_count is
       when 4 => local_count_en <= '1'; --*********************************
       when 15 => local_count_en <= '1'; --*********************************
       when others => local_count_en <= '0';
     end case;
   end if;
  end if;
 end process LOCAL_WN_EN_PROCESS;

 LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
 begin
 if ( PHASE_CLK = '0' ) then
    local_current_add <= ( OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (local_count_en = '1' ) then
      if ( local_current_add = "111" ) then
          local_current_add <= "000";
      else
       local_current_add <= local_current_add + "001";
      end if;
    end if;
 end if;
 end process LOCAL_INC_CLK_PROCESS;
```

```
  LOCAL_WN_ADD_CLK:process(MCLK)
  begin
    if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        LOCAL_WN_ADD <= local_current_add;
    end if;
  end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------
end rtl;
```

第8項　ss_u7 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u7p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;
```

```
    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u7p3;

-- ss_u7p3 Architecture Description

architecture rtl of ss_u7p3 is

-----------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-----------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-----------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-----------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-----------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-----------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
```

```
  signal next_fft11_state2   : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2   : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
-------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state   : three_states := s30;
  signal next_rd_state      : three_states := s31;

-------------------------------------------------------------------------
begin
-------------------------------------------------------------------------


-------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
   when 10 => next_rd_state <= s31;  -- ****************************
   when 12 => next_rd_state <= s32;  -- ****************************
   when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;
```

```
RD_EN_DECODE:process(present_rd_state)
begin
 case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
 end case;
end process RD_EN_DECODE;

RD_EN_STATE_REGISTER:process(MCLK, start)
begin
 if (start = '0') then
       present_rd_state <= s30;
 elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
       present_rd_state <= next_rd_state;
 end if;
end process RD_EN_STATE_REGISTER;

RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
begin
if ( start2 = '0' ) then
  RD_CONT <= ( OTHERS => '0');
elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
  RD_CONT <= rd_enable;
end if;
end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
       start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
  end if;
 end process START2_PROCESS;

 INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
  begin
   if ( START = '0' ) then -- Initializing local phase counters
      fir_count <= 0;
        fft1_count <= 0;
      fft2_count <= 0;
        spec_count <= 0;
      fir_en2 <= '0';
```

```
        fft1_en2 <= '0';
        fft2_en2 <= '0';
        spec_en2 <= '0';
      elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
        fir_en2 <= fir_en;
        fft1_en2 <= fft1_en;
        fft2_en2 <= fft2_en;
        spec_en2 <= spec_en;
-- The followings are counter for EN*
       if (fir_en = '1') then
        fir_count <= fir_count + 1;
       end if;
       if (fft1_en = '1') then
        fft1_count <= fft1_count + 1;
       end if;
       if (fft2_en = '1') then
        fft2_count <= fft2_count + 1;
       end if;
       if (spec_en = '1') then
        spec_count <= spec_count + 1;
       end if;
      end if;
   end process INIT_SEQ;

  INIT_SEQ2:process(START2,MCLK)
  begin
  if ( START2 = '0' ) then
     fir_count2 <= 0;
     fft1_count2 <= 0;
     fft2_count2 <= 0;
     spec_count2 <= 0;

  elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
     fir_count2 <=  fir_count;
     fft1_count2 <= fft1_count;
     fft2_count2 <= fft2_count;
     spec_count2 <= spec_count;
  end if;
  end process INIT_SEQ2;

-------------------------------------------------------------------------
-- ENABLE CONTROL
-------------------------------------------------------------------------
-- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
--          FIR Qch OUT ENABLE( Ch.A,B)
-- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
--          FFT 1st Qch OUT ENABLE( Ch.A,B)
-------------------------------------------------------------------------
```

```
FFT11_EN_STATE_DECODE:process(fft1_count2)
begin
next_fft11_state2 <= ss10;
case fft1_count2 is
 when 2 => next_fft11_state2 <= ss11;  -- ***************************
 when 4 => next_fft11_state2 <= ss11;  -- ***************************
 when 7 => next_fft11_state2 <= ss11;  -- ***************************
 when 9 => next_fft11_state2 <= ss11;  -- ***************************
 when OTHERS => next_fft11_state2 <= ss10;
end case;
end process FFT11_EN_STATE_DECODE;

-------------------------------------------------------------------------
FFT21_EN_STATE_DECODE:process(fft2_count2)
begin
next_fft21_state2 <= ss20;
case fft2_count2 is
 when 3 => next_fft21_state2 <= ss21;  -- ***************************
 when 8 => next_fft21_state2 <= ss21;  -- ***************************
 when OTHERS => next_fft21_state2 <= ss20;
end case;
end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
EN_STATE_REGISTER:process(MCLK, START)
begin
 if (START = '0') then
     present_fft11_state2 <= ss10;
     present_fft21_state2 <= ss20;
 elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
       if ( fft1_en = '1' ) then
         present_fft11_state2 <= next_fft11_state2;
       end if;
       if ( fft2_en = '1' ) then
         present_fft21_state2 <= next_fft21_state2;
       end if;
 end if;
end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
begin
 case present_fft11_state2 is
 when ss10 => ENABLE(0) <= '1'; -- Enable nothing
 when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
 when others => ENABLE(0) <= '1'; -- Enable nothing
```

```
 end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
   when 2 => next_fft21_state <= s11;  -- ***************************
   when 7 => next_fft21_state <= s11;  -- ***************************
   when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
        if ( FFT1_EN2 = '1' ) then
          present_fft21_state <= next_fft21_state;
        end if;
  end if;
 end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
```

```
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START)
 begin
 if ( START = '0' ) then
   FFT_SEL <= ( OTHERS => '0');
   INPUT_DATA_SEL <= ( OTHERS => '0');
   IQ_RAW_SEL <= '0';
   FIR_SEL <= '0';
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
   FFT_SEL <= mux_sel2;
   INPUT_DATA_SEL <= mux_sel;
   IQ_RAW_SEL <= mux_sel3;
   FIR_SEL <= OUTSEL;
 end if;
 end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
 begin
 if ( START2 = '0' ) then
   OUT_EN <= ( OTHERS => '1');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   OUT_EN <= ENABLE;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------

 INPUT_DATA_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (spec_en = '1' ) then
    mux_sel <= "11";
   elsif ( fft2_en = '1') then
    mux_sel <= "10";
   elsif ( fft1_en = '1') then
```

```
    mux_sel <= "10";
   else
    case fir_count is
     when 1 => mux_sel <= "00"; --*********************************
     when 12 => mux_sel <= "00"; --*********************************
     when others => mux_sel <= "01";
    end case;
   end if;
  end if;
 end process INPUT_DATA_MUX;

-------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------

 IQ_RAW_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel3 <= '0';
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fir_en = '1') then
    case fir_count is
     when 11 => mux_sel3 <= '1';
     when 22 => mux_sel3 <= '1';
     when others => mux_sel3 <= '0';
    end case;
   end if;
  end if;
 end process IQ_RAW_MUX;

-------------------------------------------------------------------------
-- FFT MUX PROCESSES ************* Each Sequencer has own procedure ********
-------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft2_en = '1') then
    if (fft2_count < 6) then --*********************************
     mux_sel2 <= "10";
    elsif (fft2_count < 11) then --*********************************
     mux_sel2 <= "11";
    end if;
   elsif ( fft1_en = '1') then --*********************************
    if (fft1_count < 3) then --*********************************
     mux_sel2 <= "01";
```

```
    elsif (fft1_count < 4) then --*********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 5) then --*********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 6) then --*********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 8) then --*********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 9) then --*********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 10) then --*********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 11) then --*********************************
     mux_sel2 <= "01";
    end if;
   end if;
  end if;
end process FFT_MUX;

-------------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
     fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fft_en = '1') then
    if    (fft1_count2 < 2 ) then --*******************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 6 ) then --*******************************
     fft_count_en <= '1';
    elsif (fft1_count2 < 7 ) then --*******************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 11 ) then --*******************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 2) then --*******************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 6 ) then --*******************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 7 ) then --*******************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 11 ) then  --*******************************
```

```
    fft_count_en <= '1';
   else
    fft_count_en <= '0';
   end if;
  end if;
 end if;
 end process FFT_WN_EN_PROCESS;

 FFT_INC_CLK_PROCESS:process(MCLK, START)
 begin
 if (START = '0') then
  fft_current_add <= (OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fft_count_en = '1' ) then
       fft_current_add <= fft_current_add + "0001";
    end if;
 end if;
 end process FFT_INC_CLK_PROCESS;

 FFT_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        FFT_WN_ADD <= fft_current_add;
   end if;
 end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 ADD_STORE_EN_PROCESS:process(fir_count2)
 begin
 if (fir_count2 = 3) then
    add_store_en <= '1';
 else
    add_store_en <= '0';
 end if;
 end process ADD_STORE_EN_PROCESS;

 RD_RAM_LD_PROCESS:process(MRST, MCLK)
 begin
 if (MRST = '1') then
  ram_initial_add <= (others => '0');
 elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
  if ( add_store_en = '1') then
   ram_initial_add <= rd_current_add(3 downto 0);
  end if;
 end if;
 end process RD_RAM_LD_PROCESS;
```

```
RD_RAM_EN_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en2 = '1') then
   if (fir_count2 < 2 ) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 11) then
    ram_count_en <= '1';
    iq_state <= '0';
   elsif (fir_count2 = 11) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 22 ) then
    ram_count_en <= '1';
    iq_state <= '1';
   else
    ram_count_en <= '0';
    iq_state <= '0';
   end if;
  end if;
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (START2 = '0') then
    rd_current_add(3 downto 0) <= ram_initial_add;
    rd_current_add(4) <= '0';
   elsif (ram_count_en = '1' ) then
    if ( rd_current_add(3 downto 0) = "1001") then
        rd_current_add(3 downto 0) <= "0000";
        rd_current_add(4) <= iq_state;
    else
      rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
        rd_current_add(4) <= iq_state;
    end if;
   end if;
 end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  RD_RAM_ADD <= rd_current_add;
 end if;
```

```
 end process RD_RAM_ADD_OUT_PROCESS;

-------------------------------------------------------------------------
-- LOCAL Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------

 LOCAL_WN_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en = '1') then
    case fir_count is
     when 4 => local_count_en <= '1'; --*********************************
     when 15 => local_count_en <= '1'; --*********************************
     when others => local_count_en <= '0';
    end case;
   end if;
  end if;
 end process LOCAL_WN_EN_PROCESS;

 LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
 begin
 if ( PHASE_CLK = '0' ) then
    local_current_add <= ( OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (local_count_en = '1' ) then
     if ( local_current_add = "111" ) then
        local_current_add <= "000";
     else
      local_current_add <= local_current_add + "001";
     end if;
    end if;
 end if;
 end process LOCAL_INC_CLK_PROCESS;

 LOCAL_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       LOCAL_WN_ADD <= local_current_add;
   end if;
 end process LOCAL_WN_ADD_CLK;

-------------------------------------------------------------------------
end rtl;
```

第9項　ss_u8 のVHDL Listing

```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
```

```
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, November, 2, 1993
--
-- Slave Sequencer for U1 test
--
-- Funtion
--
-- 1. FFT MUX Control
-- 2. INPUT MUX Control
-- 3. DATA MUX Control

-- 4. OUT_EN (3bit), FIR_SEL (2bit) Control

-- 6. I, Qch RAM Control
-- 7. FFT Wn RAM Control
-- 8. Local Wn RAM Control

-- 9. Input TRIG Control

-- MRST and START are active low signal in this module.

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Slave Sequencer Entity Description

entity ss_u8p3 is
  port(
    MCLK, PHASE_CLK: in qsim_state;
    START,MRST: in qsim_state;
    FIR_EN,FFT1_EN,FFT2_EN,SPEC_EN: in qsim_state;

    FFT_SEL, OUT_EN, INPUT_DATA_SEL: out qsim_state_vector(1 downto 0);
    FIR_SEL, IQ_RAW_SEL: out qsim_state := '0';

    RD_RAM_ADD: out qsim_state_vector(4 downto 0);
    FFT_WN_ADD: out qsim_state_vector(3 downto 0):= (OTHERS => '0');
    LOCAL_WN_ADD: out qsim_state_vector(2 downto 0):= (OTHERS => '0');
    RD_CONT: out qsim_state_vector(2 downto 0)
    );
end ss_u8p3;

-- ss_u8p3 Architecture Description

architecture rtl of ss_u8p3 is

------------------------------------------------------------------
```

```
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------
  signal start2: qsim_state;
  signal fir_en2, fft1_en2, fft2_en2, spec_en2 : qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-- This is for TRIG control.
-------------------------------------------------------------------
  type   fft21_sel_states IS (s10, s11);

  signal present_fft21_state : fft21_sel_states := s10;
  signal next_fft21_state    : fft21_sel_states := s11;

  signal local_count: integer range 0 to 3 := 0;
  signal fir_count: integer range 0 to 28 := 0;
  signal fft1_count: integer range 0 to 16 := 0;
  signal fft2_count: integer range 0 to 16 := 0;
  signal spec_count: integer range 0 to 9 := 0;
  signal OUTSEL:  qsim_state;

  signal mux_sel: qsim_state_vector(1 downto 0);
  signal mux_sel2: qsim_state_vector(1 downto 0);
  signal mux_sel3: qsim_state;

-------------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-- This is for ENABLE control.
-------------------------------------------------------------------
  type   fft11_en_states IS (ss10, ss11);
  type   fft21_en_states IS (ss20, ss21);

  signal present_fft11_state2 : fft11_en_states := ss10;
  signal next_fft11_state2    : fft11_en_states := ss11;
  signal present_fft21_state2 : fft21_en_states := ss20;
  signal next_fft21_state2    : fft21_en_states := ss21;

  signal fir_count2: integer range 0 to 28 := 0;
  signal fft1_count2: integer range 0 to 16 := 0;
  signal fft2_count2: integer range 0 to 16 := 0;
  signal spec_count2: integer range 0 to 9 := 0;
  signal ENABLE: qsim_state_vector(1 downto 0);

-------------------------------------------------------------------------
-- For local Wn address generator
-------------------------------------------------------------------------
  signal local_current_add: qsim_state_vector(2 downto 0);
  signal local_count_en : qsim_state;
```

```
--------------------------------------------------------------------------------
  signal add_store_en, ram_count_en, rd_ram_we, iq_state : qsim_state := '0';
  signal ram_initial_add : qsim_state_vector(3 downto 0) := ( others => '0');
  signal rd_current_add : qsim_state_vector(4 downto 0) := (others => '0');
  signal fft_count_en,fft_count_en2, fft_en : qsim_state;
  signal fft_current_add, fft_current_add2 : qsim_state_vector(3 downto 0):= (OTHERS=>'0');

--------------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
--------------------------------------------------------------------------------
  type   three_states IS (s30, s31, s32);
  signal rd_enable : qsim_state_vector(2 downto 0);
  signal present_rd_state    : three_states := s30;
  signal next_rd_state       : three_states := s31;

--------------------------------------------------------------------------------
begin
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- RAW DATA MONITOR ENABLE CONTROL
--------------------------------------------------------------------------------

 RD_EN_STATE_DECODE:process(fir_count2)
 begin
 next_rd_state <= s30;
 case fir_count2 is
  when 14 => next_rd_state <= s31;   -- ****************************
  when 16 => next_rd_state <= s32;   -- ****************************
  when OTHERS => next_rd_state <= s30;
 end case;
 end process RD_EN_STATE_DECODE;

 RD_EN_DECODE:process(present_rd_state)
 begin
  case present_rd_state is
  when s31 => rd_enable <= "101"; -- Enable ch.A
  when s32 => rd_enable <= "110"; -- Enable ch.B
  when others => rd_enable <= "000"; -- Enable nothing
  end case;
 end process RD_EN_DECODE;

 RD_EN_STATE_REGISTER:process(MCLK, start)
 begin
  if (start = '0') then
        present_rd_state <= s30;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        present_rd_state <= next_rd_state;
```

```
  end if;
 end process RD_EN_STATE_REGISTER;

 RD_OUTPUT_REGISTER_TRIGGER:process(MCLK, start2)
 begin
 if ( start2 = '0' ) then
   RD_CONT <= ( OTHERS => '0');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   RD_CONT <= rd_enable;
 end if;
 end process RD_OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- END OF RAW DATA MONITOR ENABLE CONTROL
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  START2_PROCESS:process(MRST, MCLK)
  begin
   if ( MRST = '1') then
        start2 <= '1';
   elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
        start2 <= START;
   end if;
  end process START2_PROCESS;

  INIT_SEQ:process(START,MCLK) -- STATE for EN* must be triggered by MCLK2.
   begin
    if ( START = '0' ) then -- Initializing local phase counters
      fir_count <= 0;
        fft1_count <= 0;
      fft2_count <= 0;
        spec_count <= 0;
      fir_en2 <= '0';
      fft1_en2 <= '0';
      fft2_en2 <= '0';
      spec_en2 <= '0';
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
      fir_en2 <= fir_en;
      fft1_en2 <= fft1_en;
      fft2_en2 <= fft2_en;
      spec_en2 <= spec_en;
-- The followings are counter for EN*
      if (fir_en = '1') then
       fir_count <= fir_count + 1;
      end if;
      if (fft1_en = '1') then
       fft1_count <= fft1_count + 1;
      end if;
```

```
      if (fft2_en = '1') then
        fft2_count <= fft2_count + 1;
      end if;
      if (spec_en = '1') then
        spec_count <= spec_count + 1;
      end if;
    end if;
  end process INIT_SEQ;

 INIT_SEQ2:process(START2,MCLK)
 begin
 if ( START2 = '0' ) then
    fir_count2 <= 0;
    fft1_count2 <= 0;
    fft2_count2 <= 0;
    spec_count2 <= 0;

 elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
    fir_count2 <=  fir_count;
    fft1_count2 <= fft1_count;
    fft2_count2 <= fft2_count;
    spec_count2 <= spec_count;
 end if;
 end process INIT_SEQ2;

 ------------------------------------------------------------------------
 -- ENABLE CONTROL
 ------------------------------------------------------------------------
 -- FFT11_EN : FIR Ich OUT ENABLE( Ch.A,B)
 --          FIR Qch OUT ENABLE( Ch.A,B)
 -- FFT21_EN : FFT 1st Ich OUT ENABLE( Ch.A,B)
 --          FFT 1st Qch OUT ENABLE( Ch.A,B)
 ------------------------------------------------------------------------

  FFT11_EN_STATE_DECODE:process(fft1_count2)
  begin
  next_fft11_state2 <= ss10;
  case fft1_count2 is
   when 2 => next_fft11_state2 <= ss11;  -- ****************************
   when 4 => next_fft11_state2 <= ss11;  -- ****************************
   when 7 => next_fft11_state2 <= ss11;  -- ****************************
   when 9 => next_fft11_state2 <= ss11;  -- ****************************
   when OTHERS => next_fft11_state2 <= ss10;
  end case;
  end process FFT11_EN_STATE_DECODE;

 ------------------------------------------------------------------------
  FFT21_EN_STATE_DECODE:process(fft2_count2)
```

```
 begin
 next_fft21_state2 <= ss20;
 case fft2_count2 is
  when 4 => next_fft21_state2 <= ss21;  -- ****************************
  when 9 => next_fft21_state2 <= ss21;  -- ****************************
  when OTHERS => next_fft21_state2 <= ss20;
 end case;
 end process FFT21_EN_STATE_DECODE;

-------------------------------------------------------------------------
 EN_STATE_REGISTER:process(MCLK, START)
 begin
  if (START = '0') then
      present_fft11_state2 <= ss10;
      present_fft21_state2 <= ss20;
  elsif ( MCLK'EVENT AND (MCLK = '0') AND (MCLK'LAST_VALUE = '1')) THEN
        if ( fft1_en = '1' ) then
          present_fft11_state2 <= next_fft11_state2;
        end if;
        if ( fft2_en = '1' ) then
          present_fft21_state2 <= next_fft21_state2;
        end if;
  end if;
 end process EN_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT11_OUTPUT_EN_DECODE:process(present_fft11_state2)
 begin
  case present_fft11_state2 is
   when ss10 => ENABLE(0) <= '1'; -- Enable nothing
   when ss11 => ENABLE(0) <= '0'; -- Enable FFT1 REAL, IMAG
   when others => ENABLE(0) <= '1'; -- Enable nothing
  end case;
 end process FFT11_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------

 FFT21_OUTPUT_EN_DECODE:process(present_fft21_state2)
 begin
  case present_fft21_state2 is
   when ss20 => ENABLE(1) <= '1'; -- Enable nothing
   when ss21 => ENABLE(1) <= '0'; -- Enable Synth REAL and IMAG
   when others => ENABLE(1) <= '1'; -- Enable nothing
  end case;
 end process FFT21_OUTPUT_EN_DECODE;

-------------------------------------------------------------------------
```

```
-- OUTSEL PROCESSES ********** Each Sequencer has own procedure *********
-------------------------------------------------------------------------

 FFT21_SEL_STATE_DECODE:process(fft1_count)
 begin
 next_fft21_state <= s10;
 case fft1_count is
  when 2 => next_fft21_state <= s11;  -- ****************************
  when 7 => next_fft21_state <= s11;  -- ****************************
  when OTHERS => next_fft21_state <= s10;
 end case;
 end process FFT21_SEL_STATE_DECODE;

-------------------------------------------------------------------------

 TRIG_STATE_REGISTER:process(MCLK, START2)

 begin
  if (START2 = '0') then
      present_fft21_state <= s10;
  elsif ( MCLK'EVENT AND (MCLK = '1') AND (MCLK'LAST_VALUE = '0')) THEN
        if ( FFT1_EN2 = '1' ) then
          present_fft21_state <= next_fft21_state;
        end if;
  end if;
 end process TRIG_STATE_REGISTER;

-------------------------------------------------------------------------

 FFT21_OUTPUT_TRIG_DECODE:process(present_fft21_state)
 begin
 case present_fft21_state is
  when s10 => OUTSEL <= '0'; -- SEL FIR CH.A
  when s11 => OUTSEL <= '1'; -- SEL FIR CH.B
 end case;
 end process FFT21_OUTPUT_TRIG_DECODE;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------

 OUTPUT_REGISTER_EN:process(MCLK, START)
 begin
 if ( START = '0' ) then
   FFT_SEL <= ( OTHERS => '0');
   INPUT_DATA_SEL <= ( OTHERS => '0');
   IQ_RAW_SEL <= '0';
   FIR_SEL <= '0';
```

```
 elsif ( MCLK'EVENT AND ( MCLK = '0' ) AND ( MCLK'LAST_VALUE = '1' )) then
   FFT_SEL <= mux_sel2;
   INPUT_DATA_SEL <= mux_sel;
   IQ_RAW_SEL <= mux_sel3;
   FIR_SEL <= OUTSEL;
 end if;
 end process OUTPUT_REGISTER_EN;

-------------------------------------------------------------------------

 OUTPUT_REGISTER_TRIGGER:process(MCLK, START2)
 begin
 if ( START2 = '0' ) then
   OUT_EN <= ( OTHERS => '1');
 elsif ( MCLK'EVENT AND ( MCLK = '1' ) AND ( MCLK'LAST_VALUE = '0' )) then
   OUT_EN <= ENABLE;
 end if;
 end process OUTPUT_REGISTER_TRIGGER;

-------------------------------------------------------------------------
-- INPUT & DATA MUX PROCESSES
-------------------------------------------------------------------------

 INPUT_DATA_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (spec_en = '1' ) then
    mux_sel <= "11";
   elsif ( fft2_en = '1') then
    mux_sel <= "10";
   elsif ( fft1_en = '1') then
    mux_sel <= "10";
   else
    case fir_count is
     when 1 => mux_sel <= "00"; --*********************************
     when 12 => mux_sel <= "00"; --*********************************
     when others => mux_sel <= "01";
    end case;
   end if;
  end if;
 end process INPUT_DATA_MUX;

-------------------------------------------------------------------------
-- IQ RAW MUX PROCESSES
-------------------------------------------------------------------------
```

```
IQ_RAW_MUX:process(START2, MCLK)
begin
 if (START2= '0') then
  mux_sel3 <= '0';
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (fir_en = '1') then
   case fir_count is
    when 11 => mux_sel3 <= '1';
    when 22 => mux_sel3 <= '1';
    when others => mux_sel3 <= '0';
   end case;
  end if;
 end if;
end process IQ_RAW_MUX;

-------------------------------------------------------------------------------
-- FFT MUX PROCESSES ************** Each Sequencer has own procedure ********
-------------------------------------------------------------------------------

 FFT_MUX:process(START2, MCLK)
 begin
  if (START2= '0') then
   mux_sel2 <= ( others => '0');
  elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (fft2_en = '1') then
     if (fft2_count < 3) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 4) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 5) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 6) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 8) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 9) then --**********************************
      mux_sel2 <= "11";
     elsif (fft2_count < 10) then --**********************************
      mux_sel2 <= "10";
     elsif (fft2_count < 11) then --**********************************
      mux_sel2 <= "11";
     end if;
    elsif ( fft1_en = '1') then --**********************************
     if (fft1_count < 3) then --**********************************
      mux_sel2 <= "01";
     elsif (fft1_count < 4) then --**********************************
      mux_sel2 <= "00";
     elsif (fft1_count < 5) then --**********************************
```

```
     mux_sel2 <= "01";
    elsif (fft1_count < 6) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 8) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 9) then --**********************************
     mux_sel2 <= "01";
    elsif (fft1_count < 10) then --**********************************
     mux_sel2 <= "00";
    elsif (fft1_count < 11) then --**********************************
     mux_sel2 <= "01";
    end if;
   end if;
  end if;
 end process FFT_MUX;

-------------------------------------------------------------------------
-- FFT Wn RAM CONTROL PROCESSES
-------------------------------------------------------------------------
 FFT_EN_PROCESS:process(fft1_en, fft2_en)
 begin
  fft_en <= fft1_en OR fft2_en;
 end process FFT_EN_PROCESS;

 FFT_WN_EN_PROCESS:process(START, MCLK)
 begin
  if (START = '0') then
     fft_count_en <= '0';
  elsif (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fft_en = '1') then
    if    (fft1_count2 < 2 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft1_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft1_count2 < 11 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 2) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 6 ) then --**********************************
     fft_count_en <= '1';
    elsif (fft2_count2 < 7 ) then --**********************************
     fft_count_en <= '0';
    elsif (fft2_count2 < 11 ) then  --**********************************
     fft_count_en <= '1';
    else
     fft_count_en <= '0';
```

```
  end if;
 end if;
end if;
end process FFT_WN_EN_PROCESS;

FFT_INC_CLK_PROCESS:process(MCLK, START)
begin
if (START = '0') then
  fft_current_add <= (OTHERS => '0');
elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
   if (fft_count_en = '1' ) then
      fft_current_add <= fft_current_add + "0001";
   end if;
end if;
end process FFT_INC_CLK_PROCESS;

FFT_WN_ADD_CLK:process(MCLK)
begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       FFT_WN_ADD <= fft_current_add;
  end if;
end process FFT_WN_ADD_CLK;

-------------------------------------------------------------------------------
-- Ich, Qch RAM CONTROL PROCESSES
-------------------------------------------------------------------------------
ADD_STORE_EN_PROCESS:process(fir_count2)
begin
 if (fir_count2 = 3) then
    add_store_en <= '1';
 else
    add_store_en <= '0';
 end if;
end process ADD_STORE_EN_PROCESS;

RD_RAM_LD_PROCESS:process(MRST, MCLK)
begin
 if (MRST = '1') then
  ram_initial_add <= (others => '0');
 elsif (MCLK = '0' and MCLK'event and MCLK'last_value = '1') then
  if ( add_store_en = '1') then
   ram_initial_add <= rd_current_add(3 downto 0);
  end if;
 end if;
end process RD_RAM_LD_PROCESS;

RD_RAM_EN_PROCESS:process(MCLK)
begin
```

```
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  if (fir_en2 = '1') then
   if (fir_count2 < 2 ) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 11) then
    ram_count_en <= '1';
    iq_state <= '0';
   elsif (fir_count2 = 11) then
    ram_count_en <= '0';
    iq_state <= '0';
   elsif (fir_count2 < 22 ) then
    ram_count_en <= '1';
    iq_state <= '1';
   else
    ram_count_en <= '0';
    iq_state <= '0';
   end if;
  end if;
 end if;
end process RD_RAM_EN_PROCESS;

RD_RAM_INC_CLK_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
  if (START2 = '0') then
   rd_current_add(3 downto 0) <= ram_initial_add;
   rd_current_add(4) <= '0';
  elsif (ram_count_en = '1' ) then
   if ( rd_current_add(3 downto 0) = "1001") then
      rd_current_add(3 downto 0) <= "0000";
      rd_current_add(4) <= iq_state;
   else
    rd_current_add(3 downto 0) <= rd_current_add(3 downto 0) + "0001";
      rd_current_add(4) <= iq_state;
   end if;
  end if;
 end if;
end process RD_RAM_INC_CLK_PROCESS;

RD_RAM_ADD_OUT_PROCESS:process(MCLK)
begin
 if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
  RD_RAM_ADD <= rd_current_add;
 end if;
end process RD_RAM_ADD_OUT_PROCESS;
```

```
-- LOCAL Wn RAM CONTROL PROCESSES
--------------------------------------------------------------------------------

 LOCAL_WN_EN_PROCESS:process(MCLK)
 begin
  if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
   if (fir_en = '1') then
    case fir_count is
     when 4 => local_count_en <= '1'; --*********************************
     when 15 => local_count_en <= '1'; --*********************************
     when others => local_count_en <= '0';
    end case;
   end if;
  end if;
 end process LOCAL_WN_EN_PROCESS;

 LOCAL_INC_CLK_PROCESS:process(MCLK, PHASE_CLK)
 begin
 if ( PHASE_CLK = '0' ) then
    local_current_add <= ( OTHERS => '0');
 elsif (MCLK'event and MCLK = '1' and MCLK'last_value = '0') then
    if (local_count_en = '1' ) then
     if ( local_current_add = "111" ) then
        local_current_add <= "000";
     else
      local_current_add <= local_current_add + "001";
     end if;
    end if;
 end if;
 end process LOCAL_INC_CLK_PROCESS;

 LOCAL_WN_ADD_CLK:process(MCLK)
 begin
   if (MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       LOCAL_WN_ADD <= local_current_add;
   end if;
 end process LOCAL_WN_ADD_CLK;

--------------------------------------------------------------------------------
end rtl;
```

第10項　ss_u10 のVHDL Listing

```
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Tue, Aug, 1, 1994
--
```

```
-- Intermidiate Register Control for U10
--
-- Funtion
--

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- Comparator Control Entity Description

entity ss_u10p5 is
  port(
     MCLK, START, MRST: in qsim_state;
     NOS: in qsim_state_vector(1 downto 0);
     SYN_REG_TRIG, SYN_IQ_SEL, PH_ROM_SEL: out qsim_state;
     SYN_RAM_WE, MC_RAM_WE, R_RAM_I_WE, R_RAM_Q_WE: out qsim_state;
     ADD_SUB, ACC_EN: out qsim_state;
     SYN_RAM_ADD, MC_RAM_ADD, ROW_RAM_ADD: out qsim_state_vector(1 downto 0);
     OUT_TRIG: out qsim_state;
     DATA_TRIG: out qsim_state_vector(2 downto 0);
     CHM_SEL: out qsim_state_vector(1 downto 0);
     FRST: out qsim_state
   );
end ss_u10p5;

-- ss_u10p5 Architecture Description

architecture rtl of ss_u10p5 is

-------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------
     signal start2: qsim_state := '1';
-------------------------------------------------------------
-- Signal declaration for MCLK sensitive signal
-------------------------------------------------------------
-------------------------------------------------------------
-- Signal declaration for MCLK2 sensitive signal
-------------------------------------------------------------

--     SYN_REG_TRIG : trig1          : MCLK
--     OUT_TRIG   : trig2            : MCLK2
--     DATA_TRIG   : trig3(2:0) : MCLK
--     SYN_IQ_SEL  : mux1              : MCLK2
--     PH_ROM_SEL  : mux2            : MCLK2
--     SYN_RAM_WE  : we1               : MCLK
--     MC_RAM_WE   : we2               : MCLK
--     R_RAM_I_WE  : we3               : MCLK
```

```
--    R_RAM_Q_WE  : we4              : MCLK
--    ADD_SUB     : adsu             : MCLK
--    ACC_EN      : acc          : MCLK
--    SYN_RAM_ADD : add1(1:0)    : MCLK2
--    MC_RAM_ADD  : add2(1:0)    : MCLK
--    ROW_RAM_ADD : add3(1:0)    : MCLK2
--    CHM_SEL     : mux3     : MCLK
--    FRST        : fr       : MCLK

    signal count, count2, istate: integer range 0 to 63 :=0;
    signal mux1, mux2, we1, we2, we3, we4, adsu, acc: qsim_state;
    signal trig1, trig2: qsim_state;
    signal add1, add2, add3: qsim_state_vector(1 downto 0);
    signal trig3: qsim_state_vector(2 downto 0);
    signal mux3: qsim_state_vector(1 downto 0);
    signal fr: qsim_state;

    type type2_states is (s10, s11);
    type type4_states is (s20, s21, s22, s23);

    signal present_trig1_state : type2_states := s10;
    signal next_trig1_state    : type2_states := s11;
    signal present_trig2_state : type2_states := s10;
    signal next_trig2_state    : type2_states := s11;
    signal present_trig3_state : type4_states := s20;
    signal next_trig3_state    : type4_states := s21;

    signal present_mux1_state  : type2_states := s10;
    signal next_mux1_state     : type2_states := s11;
    signal present_mux2_state  : type2_states := s10;
    signal next_mux2_state     : type2_states := s11;

    signal present_we1_state  : type2_states := s10;
    signal next_we1_state     : type2_states := s11;
    signal present_we2_state  : type2_states := s10;
    signal next_we2_state     : type2_states := s11;
    signal present_we3_state  : type2_states := s10;
    signal next_we3_state     : type2_states := s11;
    signal present_we4_state  : type2_states := s10;
    signal next_we4_state     : type2_states := s11;

    signal present_adsu_state  : type2_states := s10;
    signal next_adsu_state     : type2_states := s11;

    signal present_acc_state  : type2_states := s10;
    signal next_acc_state     : type2_states := s11;

    signal present_add1_state : type4_states := s20;
```

```
    signal next_add1_state   : type4_states := s21;
    signal present_add2_state : type4_states := s20;
    signal next_add2_state   : type4_states := s21;
    signal present_add3_state : type4_states := s20;
    signal next_add3_state   : type4_states := s21;

    signal present_mux3_state : type4_states := s20;
    signal next_mux3_state   : type4_states := s21;
    signal present_fr_state  : type4_states := s20;
    signal next_fr_state     : type4_states := s21;

begin
 START2_PROCESS:process(MRST, MCLK)
 begin
  if ( MRST = '1') then
      start2 <= '1';
  elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
      start2 <= START;
  end if;
 end process START2_PROCESS;

 INCREMENT_PROCESS:process(istate)
 begin
  count <= istate;
 end process INCREMENT_PROCESS;

 INIT_SEQ:process(START,MCLK) -- incremented by MCLK2 = count
 begin
    if ( START = '0' ) then -- Initializing local phase counters
       istate <= 0;
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
            istate <= count +1;
    end if;
 end process INIT_SEQ;

 INIT_SEQ2:process(start2,MCLK) -- incremented by MCLK = count2
 begin
    if ( start2 = '0' ) then
          count2 <= 0;
    elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
          count2 <=  count;
    end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- SYN TRIG
-------------------------------------------------------------------------
 SYN_TRIG_STATE_DECODE:process(count2)
```

```
 begin
 next_trig1_state <= s10;
 case count2 is
  when 5 | 9 | 13 | 17 => next_trig1_state <= s11;  -- ******************
  when OTHERS => next_trig1_state <= s10;
 end case;
 end process SYN_TRIG_STATE_DECODE;
-------------------------------------------------------------------------
 TRIG1_OUTPUT_DECODE:process(present_trig1_state)
 begin
  case present_trig1_state is
   when s11 => trig1 <= '1';   -- Trig enable
   when others => trig1 <= '0'; -- Trig nothing
  end case;
 end process TRIG1_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- OUT TRIG
-------------------------------------------------------------------------
 OUT_TRIG_STATE_DECODE:process(count)
 begin
 next_trig2_state <= s10;
 case count is
  when 23 | 26 | 29 | 32 | 35 | 38 | 41 | 44 => next_trig2_state <= s11;  -- **********
  when OTHERS => next_trig2_state <= s10;
 end case;
 end process OUT_TRIG_STATE_DECODE;
-------------------------------------------------------------------------
 TRIG2_OUTPUT_DECODE:process(present_trig2_state)
 begin
  case present_trig2_state is
   when s11 => trig2 <= '1';   -- Trig enable
   when others => trig2 <= '0'; -- Trig nothing
  end case;
 end process TRIG2_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- DATA TRIG
-------------------------------------------------------------------------
 DT_STATE_DECODE:process(count2)
 begin
 next_trig3_state <= s20;
 case count2 is
  when 5  => next_trig3_state <= s21;  -- **************************
  when 19 => next_trig3_state <= s22;  -- **************************
  when 20 => next_trig3_state <= s23;  -- **************************
  when OTHERS => next_trig3_state <= s20;
 end case;
```

```
 end process DT_STATE_DECODE;
-------------------------------------------------------------------------
 TRIG3_OUTPUT_DECODE:process(present_trig3_state)
 begin
  case present_trig3_state is
   when s21 => trig3 <= "001";
   when s22 => trig3 <= "010";
   when s23 => trig3 <= "100";
   when others => trig3 <= "000"; -- Trig nothing
  end case;
 end process TRIG3_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- SYN IQ SEL CONTROL
-------------------------------------------------------------------------

 MUX1_STATE_DECODE:process(count)
 begin
 next_mux1_state <= s10;
 case count is
   when 21|24|27|30|33|36|39|42 => next_mux1_state <= s11;  -- ***********
   when OTHERS => next_mux1_state <= s10;
 end case;
 end process MUX1_STATE_DECODE;

-------------------------------------------------------------------------
 MUX1_OUTPUT_DECODE:process(present_mux1_state)
 begin
  case present_mux1_state is
   when s11 => mux1 <= '1';
   when others => mux1 <= '0';
  end case;
 end process MUX1_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- PHASE ROM SEL CONTROL
-------------------------------------------------------------------------

 MUX2_STATE_DECODE:process(count)
 begin
 next_mux2_state <= s10;
 case count is
   when 21|23|27|29|33|35|39|41 => next_mux2_state <= s11;  -- ***********
   when OTHERS => next_mux2_state <= s10;
 end case;
 end process MUX2_STATE_DECODE;

-------------------------------------------------------------------------
```

```
MUX2_OUTPUT_DECODE:process(present_mux2_state)
begin
 case present_mux2_state is
  when s11 => mux2 <= '1';
  when others => mux2 <= '0';
 end case;
end process MUX2_OUTPUT_DECODE;

-------------------------------------------------------------------------------
-- SYN RAM WE CONTROL
-------------------------------------------------------------------------------
 WE1_STATE_DECODE:process(count2)
 begin
 next_we1_state <= s10;
 case count2 is
  when 6 | 10 | 14 | 18 => next_we1_state <= s11;  -- ***********
  when OTHERS => next_we1_state <= s10;
 end case;
 end process WE1_STATE_DECODE;
-------------------------------------------------------------------------------
 WE1_OUTPUT_DECODE:process(present_we1_state)
 begin
  case present_we1_state is
   when s11 => we1 <= '1';
   when others => we1 <= '0';
  end case;
 end process WE1_OUTPUT_DECODE;

-------------------------------------------------------------------------------
-- MC RAM WE CONTROL
-------------------------------------------------------------------------------
 WE2_STATE_DECODE:process(count2)
 begin
 next_we2_state <= s10;
 case count2 is
  when 3 | 7 | 11 | 15 => next_we2_state <= s11;  -- ***********
  when OTHERS => next_we2_state <= s10;
 end case;
 end process WE2_STATE_DECODE;
-------------------------------------------------------------------------------
 WE2_OUTPUT_DECODE:process(present_we2_state)
 begin
  case present_we2_state is
   when s11 => we2 <= '1';
   when others => we2 <= '0';
  end case;
 end process WE2_OUTPUT_DECODE;
```

```
-------------------------------------------------------------------------
-- ROW RAM Ich WE CONTROL
-------------------------------------------------------------------------
 WE3_STATE_DECODE:process(count2)
 begin
 next_we3_state <= s10;
 case count2 is
  when 24|30|36|42 => next_we3_state <= s11;  -- ************
  when OTHERS => next_we3_state <= s10;
 end case;
 end process WE3_STATE_DECODE;
-------------------------------------------------------------------------
 WE3_OUTPUT_DECODE:process(present_we3_state)
 begin
  case present_we3_state is
   when s11 => we3 <= '1';
   when others => we3 <= '0';
  end case;
 end process WE3_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- ROW RAM Qch WE CONTROL
-------------------------------------------------------------------------
 WE4_STATE_DECODE:process(count2)
 begin
 next_we4_state <= s10;
 case count2 is
  when 27|33|39|45 => next_we4_state <= s11;  -- ************
  when OTHERS => next_we4_state <= s10;
 end case;
 end process WE4_STATE_DECODE;
-------------------------------------------------------------------------
 WE4_OUTPUT_DECODE:process(present_we4_state)
 begin
  case present_we4_state is
   when s11 => we4 <= '1';
   when others => we4 <= '0';
  end case;
 end process WE4_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- ADD_SUB CONTROL
-------------------------------------------------------------------------
 ADSU_STATE_DECODE:process(count2)
 begin
 next_adsu_state <= s10;
 case count2 is
  when 22|28|34|40 => next_adsu_state <= s11;  -- ************
```

```
  when OTHERS => next_adsu_state <= s10;
 end case;
 end process ADSU_STATE_DECODE;
-------------------------------------------------------------------------------
 ADSU_OUTPUT_DECODE:process(present_adsu_state)
 begin
  case present_adsu_state is
   when s11 => adsu <= '0';
   when others => adsu <= '1';
  end case;
 end process ADSU_OUTPUT_DECODE;

-------------------------------------------------------------------------------
-- ACC_EN CONTROL
-------------------------------------------------------------------------------
 ACC_STATE_DECODE:process(count2)
 begin
 next_acc_state <= s10;
 case count2 is
  when 21|22|24|25|27|28|30|31|33|34|36|37|39|40|42|43
        => next_acc_state <= s11;  -- ************************
  when OTHERS => next_acc_state <= s10;
 end case;
 end process ACC_STATE_DECODE;
-------------------------------------------------------------------------------
 ACC_OUTPUT_DECODE:process(present_acc_state)
 begin
  case present_acc_state is
   when s11 => acc <= '0'; -- ACC ENABLE
   when others => acc <= '1'; -- ACC DISABLE
  end case;
 end process ACC_OUTPUT_DECODE;

-------------------------------------------------------------------------------
-- SYN RAM ADDRESS CONTROL
-------------------------------------------------------------------------------
 ADD1_STATE_DECODE:process(count)
 begin
 next_add1_state <= s20;
 case count is
  when 10|11|12|13|26|27|28|29|30|31 => next_add1_state <= s21;  -- ******
  when 14|15|16|17|32|33|34|35|36|37 => next_add1_state <= s22;  -- ******
  when 18|19|38|39|40|41|42|43|44|45 => next_add1_state <= s23;  -- ******
  when OTHERS => next_add1_state <= s20;
 end case;
 end process ADD1_STATE_DECODE;
-------------------------------------------------------------------------------
 ADD1_OUTPUT_DECODE:process(present_add1_state)
```

```
 begin
  case present_add1_state is
   when s21 => add1 <= "01";
   when s22 => add1 <= "10";
   when s23 => add1 <= "11";
   when others => add1 <= "00";
  end case;
 end process ADD1_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- MAX CH RAM ADDRESS CONTROL
-------------------------------------------------------------------------
 ADD2_STATE_DECODE:process(count2)
 begin
 next_add2_state <= s20;
 case count2 is
   when 6|7|8|9|25|26|27|28|29|30 => next_add2_state <= s21;  -- ******
   when 10|11|12|13|31|32|33|34|35|36 => next_add2_state <= s22;  -- ******
   when 14|15|16|17|37|38|39|40|41|42 => next_add2_state <= s23;  -- ******
   when OTHERS => next_add2_state <= s20;
 end case;
 end process ADD2_STATE_DECODE;
-------------------------------------------------------------------------
 ADD2_OUTPUT_DECODE:process(present_add2_state)
 begin
  case present_add2_state is
   when s21 => add2 <= "01";
   when s22 => add2 <= "10";
   when s23 => add2 <= "11";
   when others => add2 <= "00";
  end case;
 end process ADD2_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- ROW RAM ADDRESS CONTROL
-------------------------------------------------------------------------
 ADD3_STATE_DECODE:process(count)
 begin
 next_add3_state <= s20;
 case count is
   when 29|30|31|32|33|34|48 => next_add3_state <= s21;  -- ******
   when 35|36|37|38|39|40|49 => next_add3_state <= s22;  -- ******
   when 41|42|43|44|45|46|50 => next_add3_state <= s23;  -- ******
   when OTHERS => next_add3_state <= s20;
 end case;
 end process ADD3_STATE_DECODE;
-------------------------------------------------------------------------
 ADD3_OUTPUT_DECODE:process(present_add3_state)
```

```
 begin
  case present_add3_state is
   when s21 => add3 <= "01";
   when s22 => add3 <= "10";
   when s23 => add3 <= "11";
   when others => add3 <= "00";
  end case;
 end process ADD3_OUTPUT_DECODE;
-------------------------------------------------------------------------
-- FIXED CHANNEL MUX CONTROL
-------------------------------------------------------------------------
 MUX3_STATE_DECODE:process(count2)
 begin
 next_mux3_state <= s20;
 case count2 is
   when 5|6|7|8    => next_mux3_state <= s21;  -- ******
   when 9|10|11|12  => next_mux3_state <= s22;  -- ******
   when 13|14|15|16 => next_mux3_state <= s23;  -- ******
   when OTHERS => next_mux3_state <= s20;
 end case;
 end process MUX3_STATE_DECODE;
-------------------------------------------------------------------------
 MUX3_OUTPUT_DECODE:process(present_mux3_state)
 begin
  case present_mux3_state is
   when s21 => mux3 <= "01";
   when s22 => mux3 <= "10";
   when s23 => mux3 <= "11";
   when others => mux3 <= "00";
  end case;
 end process MUX3_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- FIXED MODE RESET CONTROL
-------------------------------------------------------------------------
 FR_STATE_DECODE:process(count2)
 begin
 next_fr_state <= s21;
 case count2 is
   when 8|9|10|11   => next_fr_state <= s21;  -- ******
   when 12|13|14|15 => next_fr_state <= s22;  -- ******
   when 16|17|18|20 => next_fr_state <= s23;  -- ******
   when OTHERS => next_fr_state <= s20;
 end case;
 end process FR_STATE_DECODE;
-------------------------------------------------------------------------
 FR_OUTPUT_DECODE:process(NOS, present_fr_state)
 begin
```

```
 case present_fr_state is
  when s20 => fr <= '0';
  when s21 => case NOS is
              when "11"   => fr <= '1';
              when others => fr <= '0';
             end case;
  when s22 => case NOS is
              when "11"   => fr <= '1';
              when "10"   => fr <= '1';
              when others => fr <= '0';
             end case;
  when s23 => case NOS is
              when "11"   => fr <= '1';
              when "10"   => fr <= '1';
              when "01"   => fr <= '1';
              when others => fr <= '0';
             end case;
  when others => fr <= '0';
  end case;
 end process FR_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- STATE REGISTER PROCESS
-------------------------------------------------------------------------
 MCLK_STATE_REGISTER:process(mclk, start2) --  Triggered by MCLK
 begin
  if (start2 = '0') then
      present_trig2_state <= s10;
      present_mux1_state <= s10;
      present_mux2_state <= s10;
      present_add1_state <= s20;
      present_add3_state <= s20;
   elsif ( MCLK'EVENT AND MCLK = '1' AND MCLK'LAST_VALUE = '0') THEN
      present_trig2_state <= next_trig2_state;
      present_mux1_state <= next_mux1_state;
      present_mux2_state <= next_mux2_state;
      present_add1_state <= next_add1_state;
      present_add3_state <= next_add3_state;
  end if;
 end process MCLK_STATE_REGISTER;

 MCLK2_STATE_REGISTER:process(mclk, start) --  Triggered by MCLK2
 begin
  if (start = '0') then
      present_trig1_state <= s10;
      present_trig3_state <= s20;
      present_we1_state <= s10;
      present_we2_state <= s10;
```

```
        present_we3_state <= s10;
        present_we4_state <= s10;
        present_adsu_state <= s10;
        present_acc_state <= s10;
        present_add2_state <= s20;
        present_mux3_state <= s20;
        present_fr_state <= s20;
    elsif ( MCLK'EVENT AND MCLK = '0' AND MCLK'LAST_VALUE = '1') THEN
        present_trig1_state <= next_trig1_state;
        present_trig3_state <= next_trig3_state;
        present_we1_state <= next_we1_state;
        present_we2_state <= next_we2_state;
        present_we3_state <= next_we3_state;
        present_we4_state <= next_we4_state;
        present_adsu_state <= next_adsu_state;
        present_acc_state <= next_acc_state;
        present_add2_state <= next_add2_state;
        present_mux3_state <= next_mux3_state;
        present_fr_state <= next_fr_state;
    end if;
  end process MCLK2_STATE_REGISTER;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------
  MCLK_OUTPUT_REGISTER:process(MCLK, start2) -- Triggered by MCLK
  begin
  if ( start2 = '0' ) then
    SYN_REG_TRIG <= '0';
    DATA_TRIG <= ( OTHERS => '0');
    SYN_RAM_WE <= '0';
    MC_RAM_WE <= '0';
    R_RAM_I_WE <= '0';
    R_RAM_Q_WE <=  '0';
    ADD_SUB <= '0';
    ACC_EN <= '0';
    MC_RAM_ADD <= ( OTHERS => '0');
    CHM_SEL <= ( OTHERS => '0');
    FRST <= '0';
  elsif ( MCLK'EVENT AND MCLK = '1' AND MCLK'LAST_VALUE = '0' ) then
    SYN_REG_TRIG <= trig1;
    DATA_TRIG <= trig3;
    SYN_RAM_WE <= we1;
    MC_RAM_WE <= we2;
    R_RAM_I_WE <= we3;
    R_RAM_Q_WE <= we4;
    ADD_SUB <= adsu;
    ACC_EN <= acc;
```

```
    MC_RAM_ADD <= add2;
    CHM_SEL <= mux3;
    FRST <= fr;
  end if;
  end process MCLK_OUTPUT_REGISTER;

  MCLK2_OUTPUT_REGISTER:process(MCLK, start) -- Triggered by MCLK2
   begin
   if ( start = '0' ) then
    OUT_TRIG <= '0';
    SYN_IQ_SEL <= '0';
    PH_ROM_SEL <= '0';
    SYN_RAM_ADD <= ( OTHERS => '0');
    ROW_RAM_ADD <= ( OTHERS => '0');
   elsif ( MCLK'EVENT AND MCLK = '0' AND MCLK'LAST_VALUE = '1' ) then
    OUT_TRIG <= trig2;
    SYN_IQ_SEL <= mux1;
    PH_ROM_SEL <= mux2;
    SYN_RAM_ADD <= add1;
    ROW_RAM_ADD <= add3;
   end if;
  end process MCLK2_OUTPUT_REGISTER;

-------------------------------------------------------------------------
-- End of rtl.
-------------------------------------------------------------------------
end rtl;
```

第11項　sel_contのVHDL Listing

```
-- Copyright(c) 1995 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Wed. 30. Aug. 1995
--
-- IIR filter controller for Amplitude selector
--
-- Funtion
--

library mgc_portable;
use mgc_portable.qsim_logic.all;

-- tloop_count Entity Description

entity sel_cont is
  port(
      MCLK: in qsim_state;
```

```
        START: in qsim_state;
        MRST: in qsim_state;
      ACC_EN: out qsim_state;
      ADD_SUB: out qsim_state;
      LOAD: out qsim_state;
        IN_SEL: out qsim_state_vector(1 downto 0);
        AB_SEL: out qsim_state;
        OUT_TRIG: out qsim_state_vector( 1 downto 0)
    );
end sel_cont;

-- sel_cont Architecture Description

architecture rtl of sel_cont is

-------------------------------------------------------------------
-- Signal declaration for STATE GENERATOR
-------------------------------------------------------------------

--      OUT_TRIG    : trig1              : MCLK
--      IN_SEL      : mux1       : MCLK2
--      AB_SEL      : mux2         : MCLK2
--      LOAD        : ld         : MCLK2
--      ADD_SUB     : adsu       : MCLK2
--      ACC_EN      : acc        : MCLK2

    signal start2: qsim_state;
    signal count, count2, istate: integer range 0 to 15;
    signal trig1: qsim_state_vector(1 downto 0);
    signal mux1: qsim_state_vector(1 downto 0);
    signal mux2, ld, adsu, acc: qsim_state;

    type type2_states is (s10, s11);
    type type3_states is (s30, s31, s32);
    type type4_states is (s20, s21, s22, s23);

    signal present_trig1_state : type3_states := s30;
    signal next_trig1_state    : type3_states := s31;

    signal present_mux1_state  : type4_states := s20;
    signal next_mux1_state     : type4_states := s21;
    signal present_mux2_state  : type2_states := s10;
    signal next_mux2_state     : type2_states := s11;

    signal present_adsu_state  : type2_states := s10;
    signal next_adsu_state     : type2_states := s11;

    signal present_acc_state  : type2_states := s10;
```

```
    signal next_acc_state    : type2_states := s11;

    signal present_ld_state  : type2_states := s10;
    signal next_ld_state     : type2_states := s11;

begin
 START2_PROCESS:process(MRST, MCLK)
  begin
   if ( MRST = '1') then
       start2 <= '1';
   elsif ( MCLK'event and MCLK = '0' and MCLK'last_value = '1') then
       start2 <= START;
   end if;
  end process START2_PROCESS;

 INCREMENT_PROCESS:process(istate)
 begin
  count <= istate;
 end process INCREMENT_PROCESS;

 INIT_SEQ:process(START,MCLK) -- incremented by MCLK2 = count
 begin
    if ( START = '0' ) then -- Initializing local phase counters
       istate <= 0;
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
          if (istate = 15 ) then
                null;
       else
             istate <= count +1;
       end if;
    end if;
 end process INIT_SEQ;

 INIT_SEQ2:process(start2,MCLK) -- incremented by MCLK = count2
 begin
    if ( start2 = '0' ) then
          count2 <= 0;
    elsif (MCLK'event and (MCLK = '1') and (MCLK'last_value = '0')) then
          count2 <=  count;
    end if;
 end process INIT_SEQ2;

-------------------------------------------------------------------------
-- OUT TRIG
-------------------------------------------------------------------------
 TRIG1_STATE_DECODE:process(count2)
 begin
 next_trig1_state <= s30;
```

```
 case count2 is
  when 4 => next_trig1_state <= s31;  -- ***************************
  when 8 => next_trig1_state <= s32;  -- ***************************
  when OTHERS => next_trig1_state <= s30;
 end case;
 end process TRIG1_STATE_DECODE;
-------------------------------------------------------------------------
 TRIG1_OUTPUT_DECODE:process(present_trig1_state)
 begin
  case present_trig1_state is
  when s31 => trig1 <= "01";
  when s32 => trig1 <= "10";
  when others => trig1<= "00"; -- Trig nothing
  end case;
 end process TRIG1_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- INPUT SEL CONTROL
-------------------------------------------------------------------------

 MUX1_STATE_DECODE:process(count)
 begin
 next_mux1_state <= s20;
 case count is
  when 5|6 => next_mux1_state <= s21;  -- ************
  when 3|7 => next_mux1_state <= s22;  -- ************
  when 4|8 => next_mux1_state <= s23;  -- ************
  when OTHERS => next_mux1_state <= s20;
 end case;
 end process MUX1_STATE_DECODE;

-------------------------------------------------------------------------
 MUX1_OUTPUT_DECODE:process(present_mux1_state)
 begin
  case present_mux1_state is
  when s21 => mux1 <= "01";
  when s22 => mux1 <= "10";
  when s23 => mux1 <= "11";
  when others => mux1 <= "00";
  end case;
 end process MUX1_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- AB SEL CONTROL
-------------------------------------------------------------------------

 MUX2_STATE_DECODE:process(count)
 begin
```

```
 next_mux2_state <= s10;
 case count is
  when 5|6|7|8 => next_mux2_state <= s11;  -- ***********
  when OTHERS => next_mux2_state <= s10;
 end case;
 end process MUX2_STATE_DECODE;

-------------------------------------------------------------------------
 MUX2_OUTPUT_DECODE:process(present_mux2_state)
 begin
  case present_mux2_state is
   when s11 => mux2 <= '1';
   when others => mux2 <= '0';
  end case;
 end process MUX2_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- ADD_SUB CONTROL
-------------------------------------------------------------------------
 ADSU_STATE_DECODE:process(count)
 begin
 next_adsu_state <= s10;
 case count is
  when 3|7 => next_adsu_state <= s11;  -- ***********
  when OTHERS => next_adsu_state <= s10;
 end case;
 end process ADSU_STATE_DECODE;
-------------------------------------------------------------------------
 ADSU_OUTPUT_DECODE:process(present_adsu_state)
 begin
  case present_adsu_state is
   when s11 => adsu <= '0';
   when others => adsu <= '1';
  end case;
 end process ADSU_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- ACC_EN CONTROL
-------------------------------------------------------------------------
 ACC_STATE_DECODE:process(count)
 begin
 next_acc_state <= s10;
 case count is
  when 2|3|4|6|7|8 => next_acc_state <= s11;  -- *************************
  when OTHERS => next_acc_state <= s10;
 end case;
 end process ACC_STATE_DECODE;
-------------------------------------------------------------------------
```

```
ACC_OUTPUT_DECODE:process(present_acc_state)
 begin
  case present_acc_state is
   when s11 => acc <= '1'; -- ACC ENABLE
   when others => acc <= '0'; -- ACC DISABLE
  end case;
 end process ACC_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- LOAD CONTROL
-------------------------------------------------------------------------
 LOAD_STATE_DECODE:process(count)
  begin
  next_ld_state <= s10;
  case count is
   when 1 | 5 => next_ld_state <= s11;   -- *************************
   when OTHERS => next_ld_state <= s10;
  end case;
  end process LOAD_STATE_DECODE;
-------------------------------------------------------------------------
 LOAD_OUTPUT_DECODE:process(present_ld_state)
  begin
   case present_ld_state is
    when s11 => ld <= '1'; -- LOAD
    when others => ld <= '0'; -- NOT LOAD
   end case;
  end process LOAD_OUTPUT_DECODE;

-------------------------------------------------------------------------
-- STATE REGISTER PROCESS
-------------------------------------------------------------------------
 MCLK_STATE_REGISTER:process(mclk, start2) --  Triggered by MCLK
  begin
   if (start2 = '0') then
       present_mux1_state <= s20;
       present_mux2_state <= s10;
       present_adsu_state <= s10;
       present_acc_state <= s10;
       present_ld_state <= s10;
    elsif ( MCLK'EVENT AND MCLK = '1' AND MCLK'LAST_VALUE = '0') THEN
       present_mux1_state <= next_mux1_state;
       present_mux2_state <= next_mux2_state;
       present_adsu_state <= next_adsu_state;
       present_acc_state <= next_acc_state;
       present_ld_state <= next_ld_state;
   end if;
  end process MCLK_STATE_REGISTER;
```

```
MCLK2_STATE_REGISTER:process(mclk, start) --  Triggered by MCLK2
 begin
  if (start = '0') then
      present_trig1_state <= s30;
   elsif ( MCLK'EVENT AND MCLK = '0' AND MCLK'LAST_VALUE = '1') THEN
      present_trig1_state <= next_trig1_state;
  end if;
 end process MCLK2_STATE_REGISTER;

-------------------------------------------------------------------------
-- TRIGGERED BY MCLK, MCLK2 PROCESS = FINAL OUTPUT
-------------------------------------------------------------------------
MCLK_OUTPUT_REGISTER:process(MCLK, start2) -- Triggered by MCLK
 begin
 if ( start2 = '0' ) then
   OUT_TRIG <= "00";
 elsif ( MCLK'EVENT AND MCLK = '1' AND MCLK'LAST_VALUE = '0' ) then
   OUT_TRIG <= trig1;
end if;
end process MCLK_OUTPUT_REGISTER;

MCLK2_OUTPUT_REGISTER:process(MCLK, start) -- Triggered by MCLK2
 begin
 if ( start = '0' ) then
   IN_SEL <= "00";
   AB_SEL <= '0';
   ADD_SUB <= '0';
   ACC_EN <= '0';
   LOAD <= '0';
 elsif ( MCLK'EVENT AND MCLK = '0' AND MCLK'LAST_VALUE = '1' ) then
   IN_SEL <= mux1;
   AB_SEL <= mux2;
   ADD_SUB <= adsu;
   ACC_EN <= acc;
   LOAD <= ld;
 end if;
end process MCLK2_OUTPUT_REGISTER;

-------------------------------------------------------------------------
-- End of rtl.
-------------------------------------------------------------------------
end rtl;
```

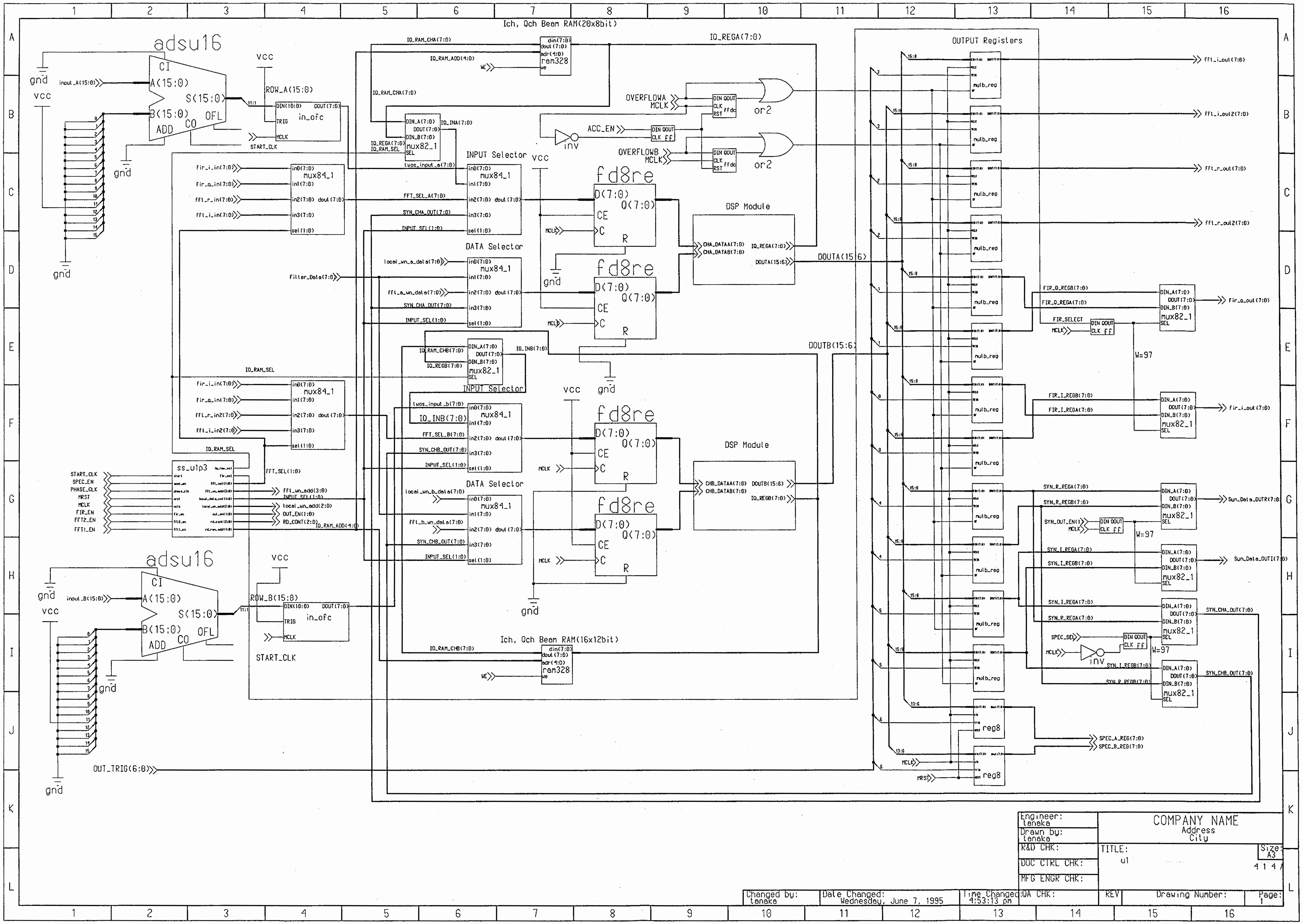
第12項　cpu_contのVHDL Listing

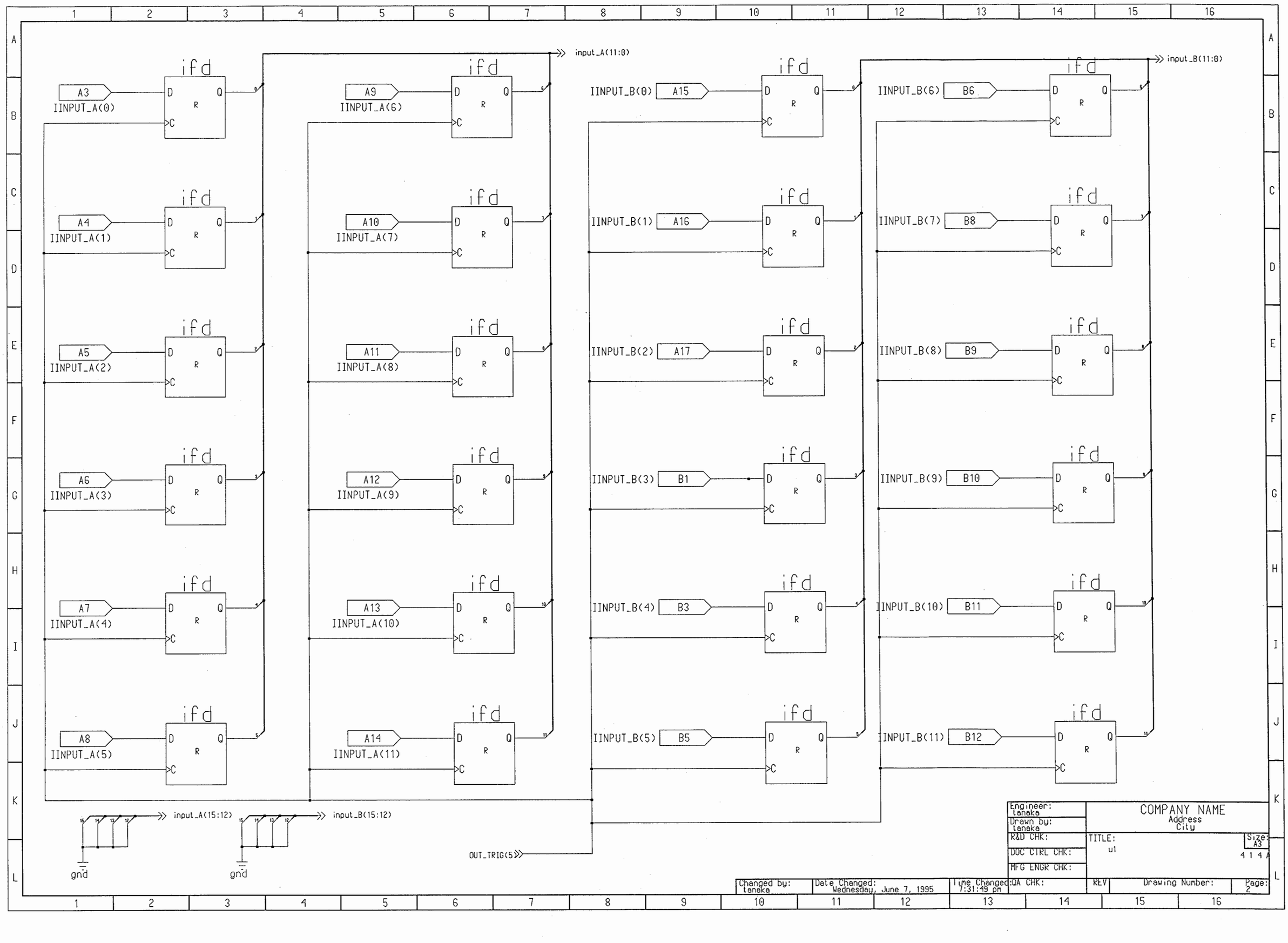
```
--
-- Copyright(c) 1993 ATR Optical and Radio Communications Research Laboratories
-- All rights reserved
```

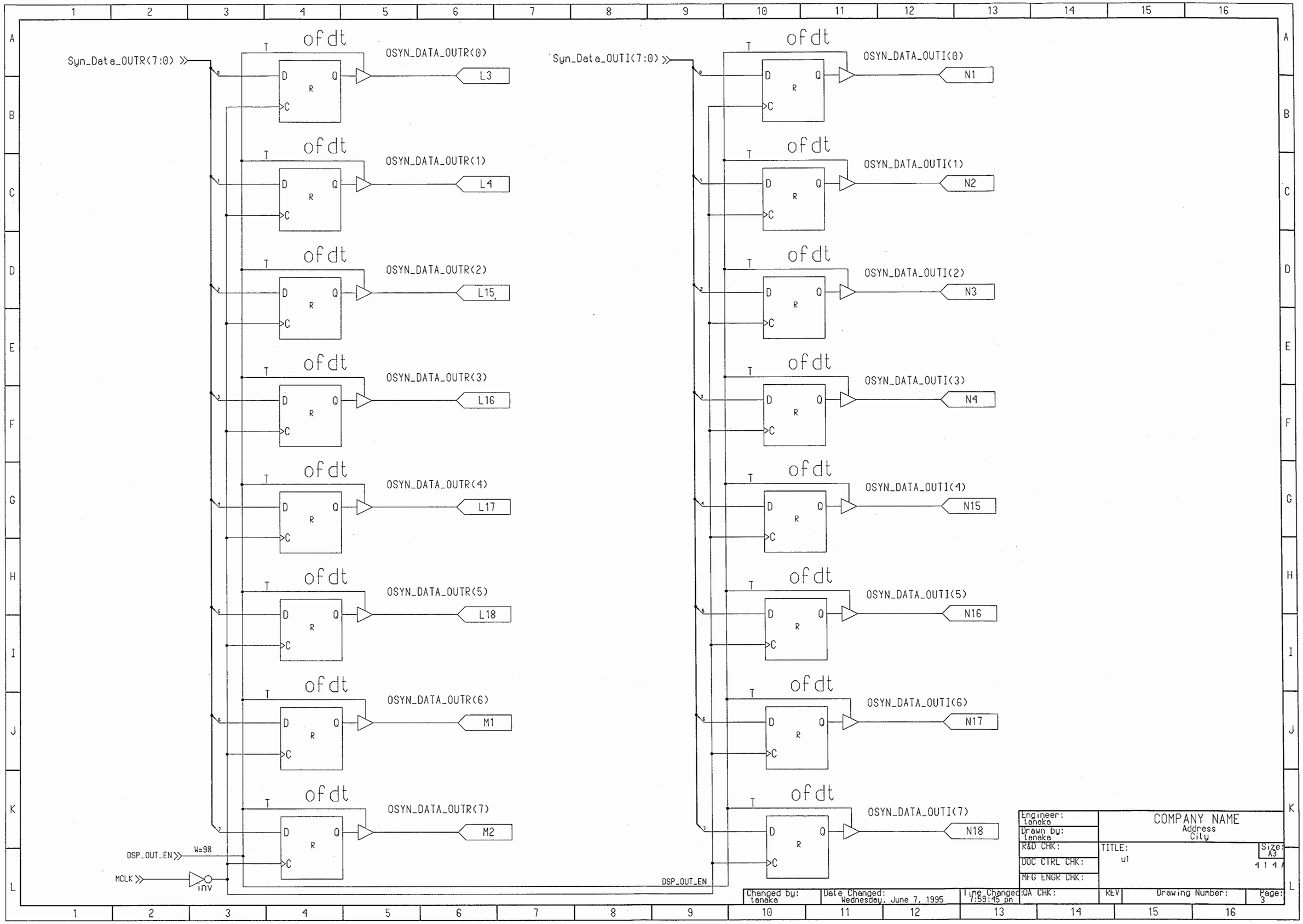
```
-- Written by Toyohisa Tanaka @Radio Communications Department
-- Rev.A Mon, August, 15, 1993
--
-- Funtion
--
-- STATE GENERATOR with MCLK/MCLK2
--
-- UART control
--
library mgc_portable;
use mgc_portable.qsim_logic.all;

-- MAIN Sequencer Entity Description

entity cpu_cont is
  port(
    MCLK, RW, DS: in qsim_state;
    DBEN, LATCH: out qsim_state := '1';
    DTACK: out qsim_state := '0'
  );
end cpu_cont;

-- cpu_cont Architecture Description
-- Moore Machine Design Model

architecture rtl of cpu_cont is

-----------------------------------------------------------------
-- Signal declaration for STATE GENERATOROR
-----------------------------------
  signal count_en, lrw : qsim_state :='0';
  SIGNAL istate, count : integer range 0 to 9 :=0;
-----------------------------------------------------------------
begin
-----------------------------------------------------------------

 INCREMENT_PROCESS:process(istate)
 begin
  count <= istate;
 end process INCREMENT_PROCESS;

 INIT_SEQ:process(count_en, MCLK) -- incremented by MCLK2 = count
 begin
    if ( count_en = '0' ) then -- When the Data  Strobe is unactivated.
       istate <= 0;
    elsif (MCLK'event and (MCLK = '0') and (MCLK'last_value = '1')) then
              istate <= count +1;
    end if;
```

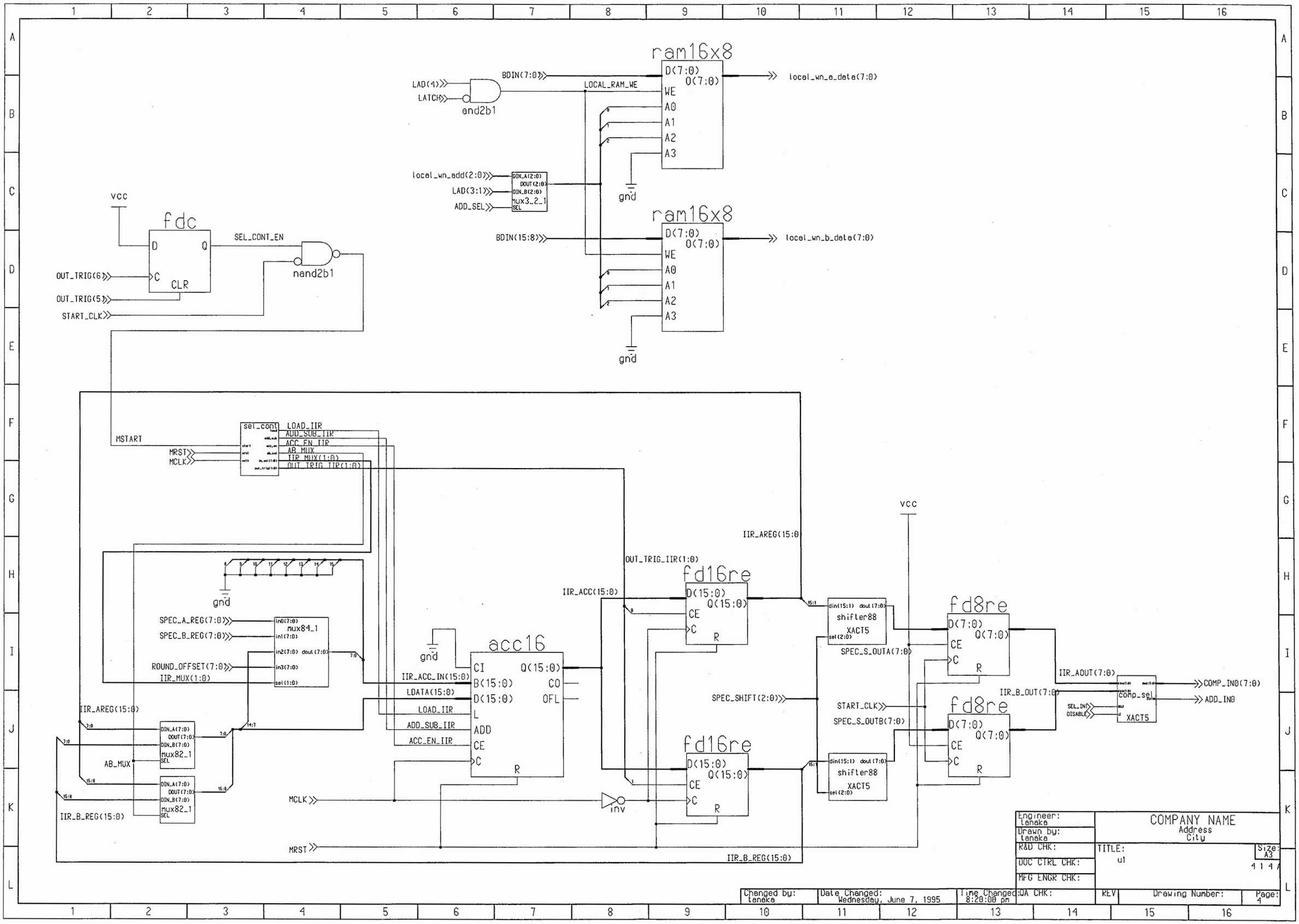
```
end process INIT_SEQ;

DS_TRIG_PROCESS:PROCESS(count, DS)
begin
 IF (count = 8) THEN
  count_en <= '0';
  lrw <= '0';
 ELSIF (DS'EVENT and DS = '1' and DS'last_value = '0') THEN
  count_en <= '1';
  lrw <= RW;
 END IF;
end PROCESS DS_TRIG_PROCESS;

DBEN_PROCESS:PROCESS(MCLK)
begin
 IF (MCLK'EVENT and MCLK='1' and MCLK'Last_value='0') then
  CASE lrw IS
   WHEN '1' =>
    CASE count IS
      WHEN 1 => DBEN <= '0';
            DTACK <= '0';
            LATCH <= '1';
      WHEN 2 => DBEN <= '0';
            DTACK <= '0'; -- Changed 95/12/19 1 to 0
            LATCH <= '1';
      WHEN 3 => DBEN <= '0';
            DTACK <= '0'; -- Changed 95/12/19 1 to 0
            LATCH <= '1';
      WHEN 4 => DBEN <= '0';
            DTACK <= '1';
            LATCH <= '1';
      WHEN 5 => DBEN <= '0';
            DTACK <= '1';
            LATCH <= '1';
      WHEN 6 => DBEN <= '1';
            DTACK <= '1';
            LATCH <= '1';
      WHEN OTHERS => DBEN <= '1';
            DTACK <= '0';
            LATCH <= '1';
    END CASE;                -- count
   WHEN '0' =>
    CASE count IS
      WHEN 1 => DBEN <= '0';
            LATCH <= '1';
            DTACK <= '0';
      WHEN 2 => DBEN <= '0';
            LATCH <= '0';
```

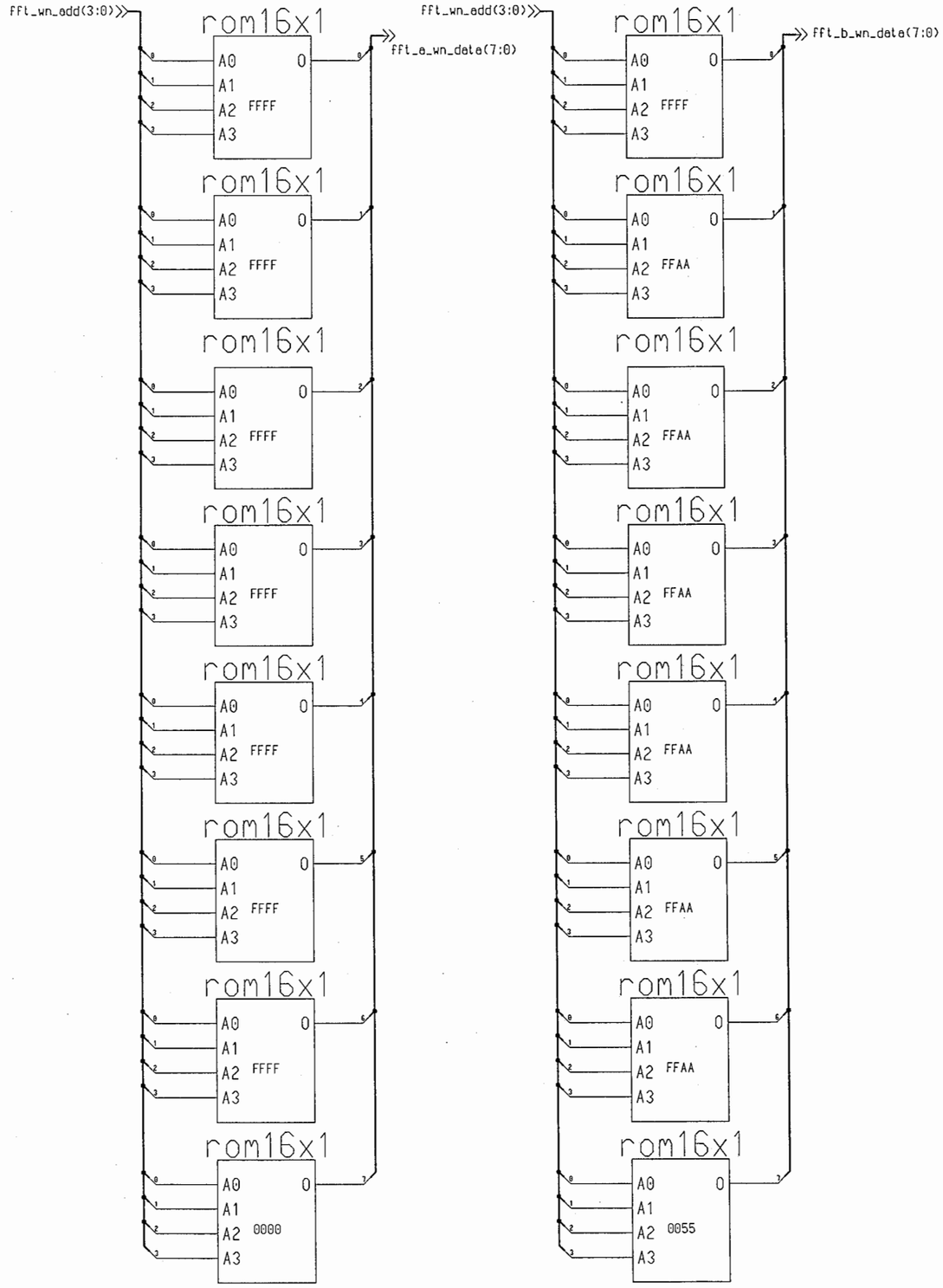
```
            DTACK <= '0';
        WHEN 3 => DBEN <= '0';
            LATCH <= '0';
            DTACK <= '0';
        WHEN 4 => DBEN <= '0';
            LATCH <= '0';
            DTACK <= '1';
        WHEN 5 => DBEN <= '0';
            LATCH <= '1';
            DTACK <= '1';
        WHEN 6 => DBEN <= '0';
            LATCH <= '1';
            DTACK <= '1';
        WHEN 7 => DBEN <= '1';
            LATCH <= '1';
            DTACK <= '1';
        WHEN OTHERS => DBEN <= '1';
            LATCH <= '1';
            DTACK <= '0';
    END CASE;                -- count
   WHEN OTHERS => NULL;
                END CASE;                                          -- RW
end if;
end PROCESS DBEN_PROCESS;

-------------------------------------------------------------------------
-- END OF ARCHITECTURE
-------------------------------------------------------------------------
end rtl;
```


Ich, Qch Beem RAM(20x8bit)
IO_REGA(7:0)
OUTPUT Registers
adsu16
CI
A(15:0)
B(15:0)
S(15:0)
OFL
ADD
CO
in_ofc
ROW_A(15:0)
START_CLK
INPUT Selector
mux84_1
mux82_1
ram328
OVERFLOWA
MCLK
ACC_EN
inv
OVERFLOWB
or2
fd8re
D(7:0)
Q(7:0)
CE
C
R
DSP Module
DATA Selector
DOUTA(15:6)
DOUTB(15:6)
W=97
mulb_reg
fft_i_out(7:0)
fft_i_out2(7:0)
fft_r_out(7:0)
fft_r_out2(7:0)
fir_q_out(7:0)
fir_i_out(7:0)
Syn_Data_OUTR(7:0)
Syn_Data_OUTI(7:0)
SYN_CHA_OUT(7:0)
SYN_CHB_OUT(7:0)
SPEC_A_REG(7:0)
SPEC_B_REG(7:0)
reg8
ss_u1p3
START_CLK
SPEC_EN
PHASE_CLK
MRST
MCLK
FIR_EN
FFT2_EN
FFT1_EN
ROW_B(15:0)
Ich, Qch Beem RAM(16x12bit)
OUT_TRIG(6:0)
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE: u1
Size: A3
REV
Drawing Number:
Page: 1
Changed by: tanaka
Date Changed: Wednesday, June 7, 1995
Time Changed: 4:53:13 pm


input_A(11:0)
input_B(11:0)
A3
IINPUT_A(0)
A4
IINPUT_A(1)
A5
IINPUT_A(2)
A6
IINPUT_A(3)
A7
IINPUT_A(4)
A8
IINPUT_A(5)
A9
IINPUT_A(6)
A10
IINPUT_A(7)
A11
IINPUT_A(8)
A12
IINPUT_A(9)
A13
IINPUT_A(10)
A14
IINPUT_A(11)
IINPUT_B(0)
A15
IINPUT_B(1)
A16
IINPUT_B(2)
A17
IINPUT_B(3)
B1
IINPUT_B(4)
B3
IINPUT_B(5)
B5
IINPUT_B(6)
B6
IINPUT_B(7)
B8
IINPUT_B(8)
B9
IINPUT_B(9)
B10
IINPUT_B(10)
B11
IINPUT_B(11)
B12
ifd
D
Q
R
C
input_A(15:12)
input_B(15:12)
gnd
OUT_TRIG(5)
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
ui
Size: A3
DA CHK:
REV
Drawing Number:
Page: 2
Changed by: tanaka
Date Changed: Wednesday, June 7, 1995
Time Changed: 7:31:49 pm

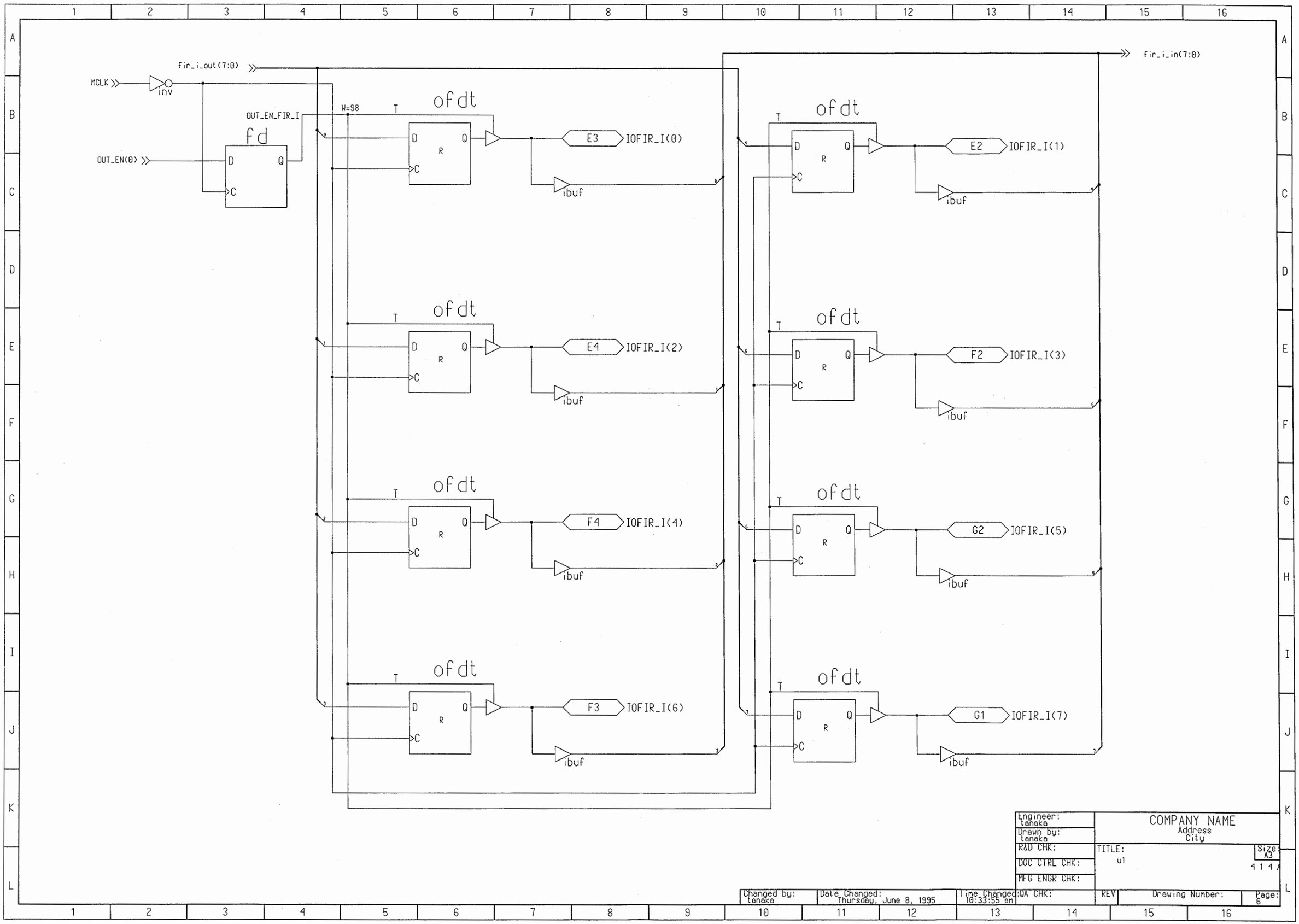

Syn_Data_OUTR(7:0)
Syn_Data_OUTI(7:0)
ofdt
T
D
Q
R
C
OSYN_DATA_OUTR(0)
OSYN_DATA_OUTR(1)
OSYN_DATA_OUTR(2)
OSYN_DATA_OUTR(3)
OSYN_DATA_OUTR(4)
OSYN_DATA_OUTR(5)
OSYN_DATA_OUTR(6)
OSYN_DATA_OUTR(7)
OSYN_DATA_OUTI(0)
OSYN_DATA_OUTI(1)
OSYN_DATA_OUTI(2)
OSYN_DATA_OUTI(3)
OSYN_DATA_OUTI(4)
OSYN_DATA_OUTI(5)
OSYN_DATA_OUTI(6)
OSYN_DATA_OUTI(7)
L3
L4
L15
L16
L17
L18
M1
M2
N1
N2
N3
N4
N15
N16
N17
N18
DSP_OUT_EN
W=98
MCLK
INV
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
Changed by: tanaka
Date Changed: Wednesday, June 7, 1995
Time Changed: 7:59:45 pm
QA CHK:
REV
Drawing Number:
Page: 3

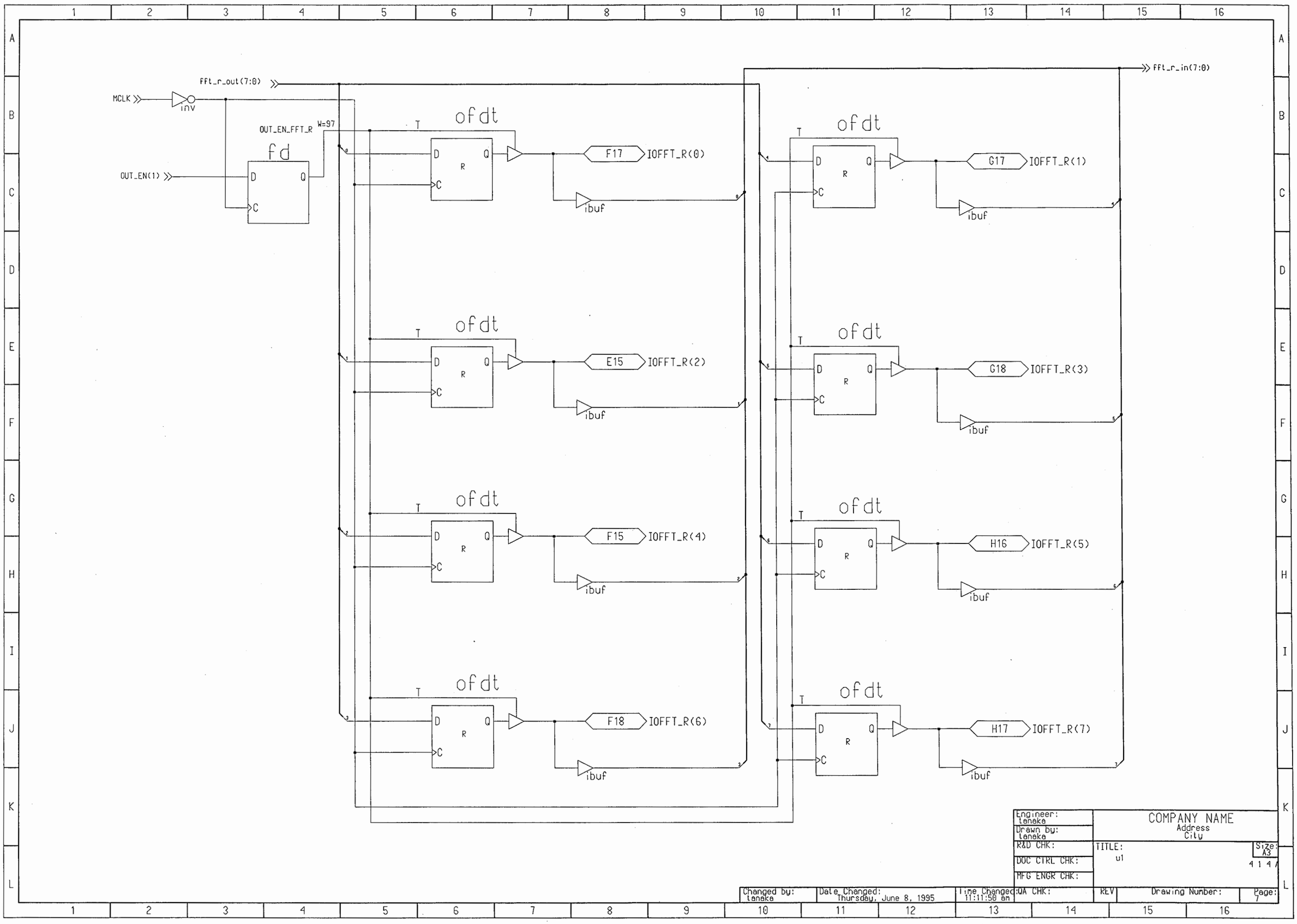

ram16x8
D(7:0)
O(7:0)
WE
A0
A1
A2
A3
BDIN(7:0)
LAD(4)
LATCH
and2b1
LOCAL_RAM_WE
local_wn_e_data(7:0)
gnd
local_wn_add(2:0)
LAD(3:1)
ADD_SEL
DIN_A(2:0)
DOUT(2:0)
DIN_B(2:0)
MUX3_2_1
SEL
BDIN(15:8)
local_wn_b_data(7:0)
vcc
fdc
D
Q
C
CLR
SEL_CONT_EN
nand2b1
OUT_TRIG(6)
OUT_TRIG(5)
START_CLK
sel_cont
LOAD_IIR
ADD_SUB_IIR
ACC_EN_IIR
AB_MUX
TIR_MUX(1:0)
OUT_TRIG_IIR(1:0)
MSTART
MRST
MCLK
IIR_AREG(15:0)
IIR_ACC(15:0)
IIR_ACC_IN(15:0)
IIR_B_REG(15:0)
SPEC_A_REG(7:0)
SPEC_B_REG(7:0)
ROUND_OFFSET(7:0)
IIR_MUX(1:0)
mux84_1
mux82_1
acc16
CI
B(15:0)
D(15:0)
L
ADD
CE
R
Q(15:0)
CO
OFL
LDATA(15:0)
fd16re
fd8re
shifter88
XACT5
SPEC_S_OUTA(7:0)
SPEC_S_OUTB(7:0)
SPEC_SHIFT(2:0)
IIR_AOUT(7:0)
IIR_B_OUT(7:0)
comp_sel
SEL_IN
DISABLE
COMP_IN0(7:0)
ADD_IN0
inv
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
Changed by: taneka
Date Changed: Wednesday, June 7, 1995
Time Changed: 8:20:00 pm
REV
Drawing Number:
Page: 4

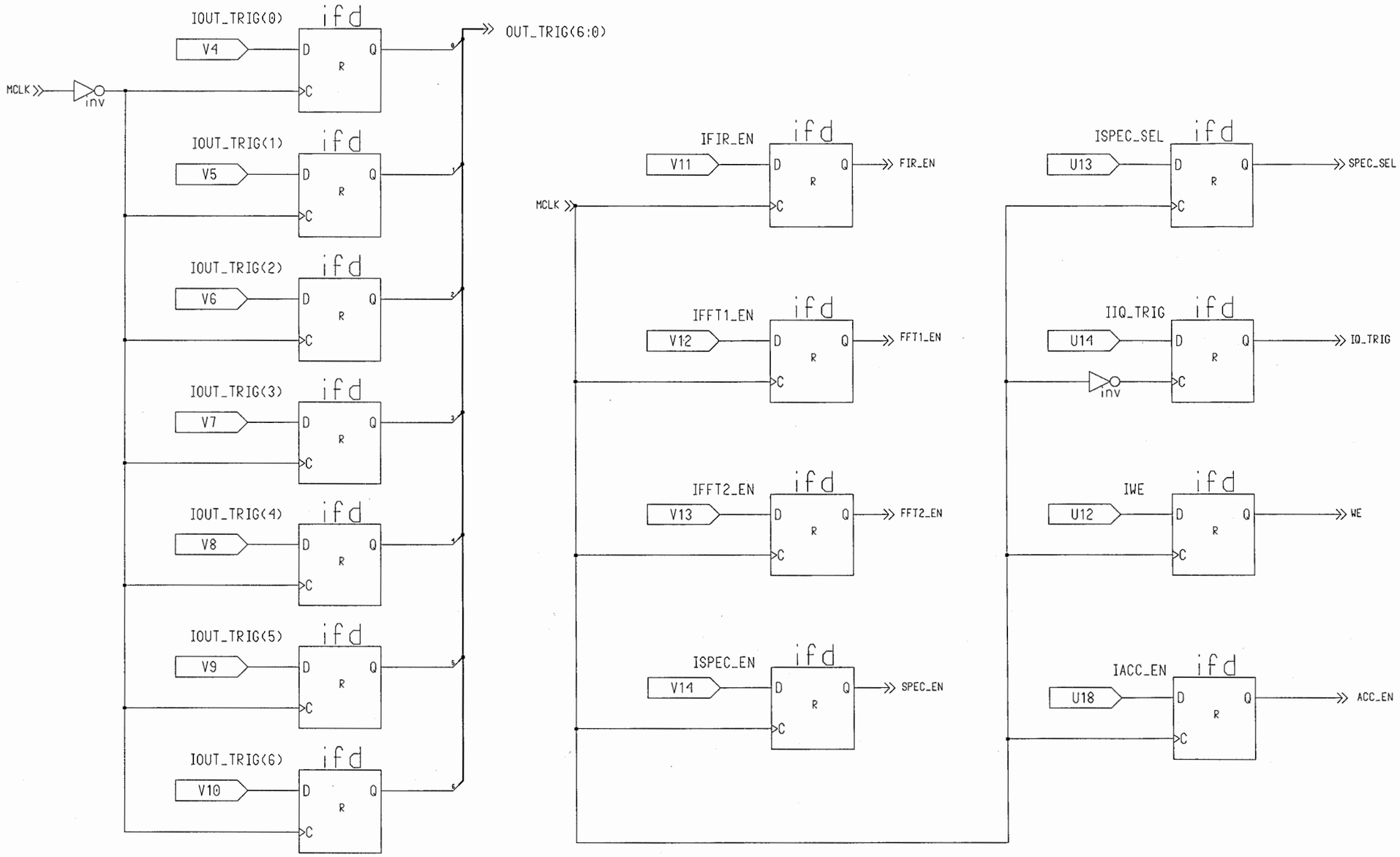


Engineer: taneke	COMPANY NAME
Drawn by: taneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u1
MFG ENGR CHK:	Size: A3
QA CHK:	4 1 4 A

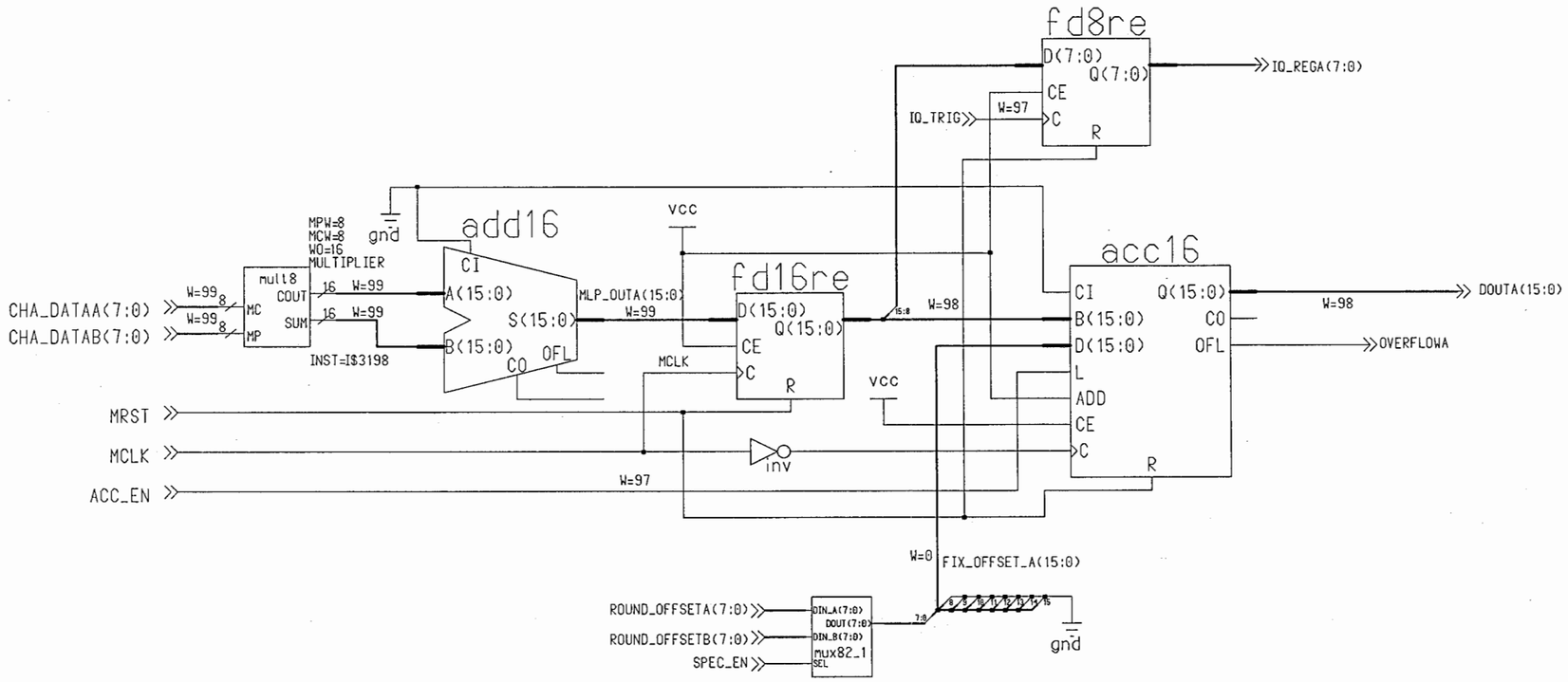
Changed by:	Date Changed:	Time Changed:	REV	Drawing Number:	Page:
taneke	Thursday, June 8, 1995	10:33:35 am			5


fir_i_out(7:0)
fir_i_in(7:0)
MCLK
inv
OUT_EN_FIR_I
OUT_EN(0)
fd
D
Q
C
W=98
T
ofdt
R
E3
IOFIR_I(0)
E2
IOFIR_I(1)
E4
IOFIR_I(2)
F2
IOFIR_I(3)
F4
IOFIR_I(4)
G2
IOFIR_I(5)
F3
IOFIR_I(6)
G1
IOFIR_I(7)
ibuf
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
4 1 4 A
Changed by: tanaka
Date Changed: Thursday, June 8, 1995
Time Changed: 10:33:55 am
QA CHK:
REV
Drawing Number:
Page: 6


FFt_r_out(7:0)
FFt_r_in(7:0)
MCLK
inv
OUT_EN_FFT_R
W=97
OUT_EN(1)
fd
D
Q
C
ofdt
T
R
F17
IOFFT_R(0)
G17
IOFFT_R(1)
E15
IOFFT_R(2)
G18
IOFFT_R(3)
F15
IOFFT_R(4)
H16
IOFFT_R(5)
F18
IOFFT_R(6)
H17
IOFFT_R(7)
ibuf
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
Changed by: tanaka
Date Changed: Thursday, June 8, 1995
Time Changed: 11:11:50 am
QA CHK:
REV
Drawing Number:
Page: 7

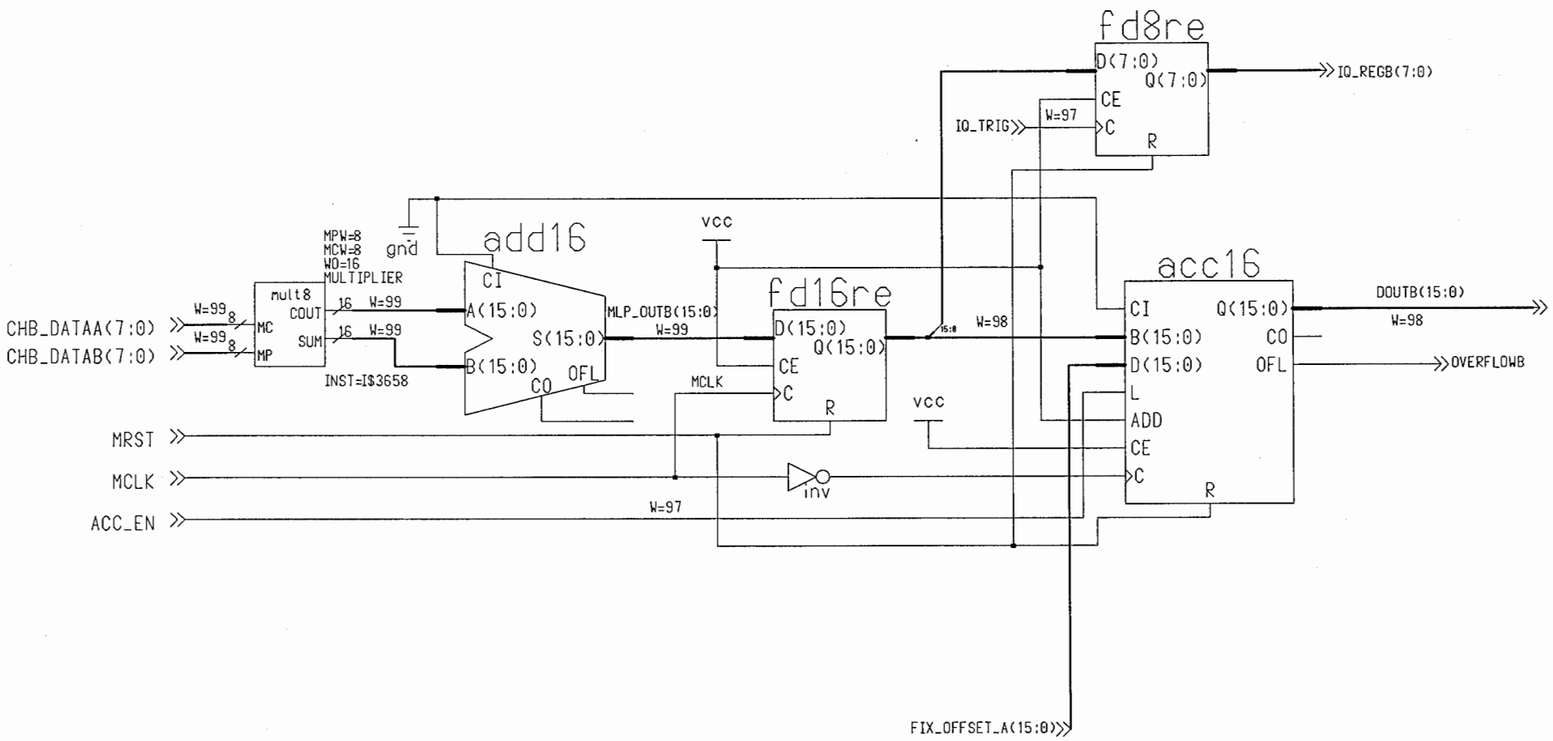


Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u1
MFG ENGR CHK:	Size: A3
Changed by: tanaka	Date Changed: Thursday, June 8, 1995
Time Changed: 11:38:41 am	QA CHK:
REV	Drawing Number:
Page: 8	

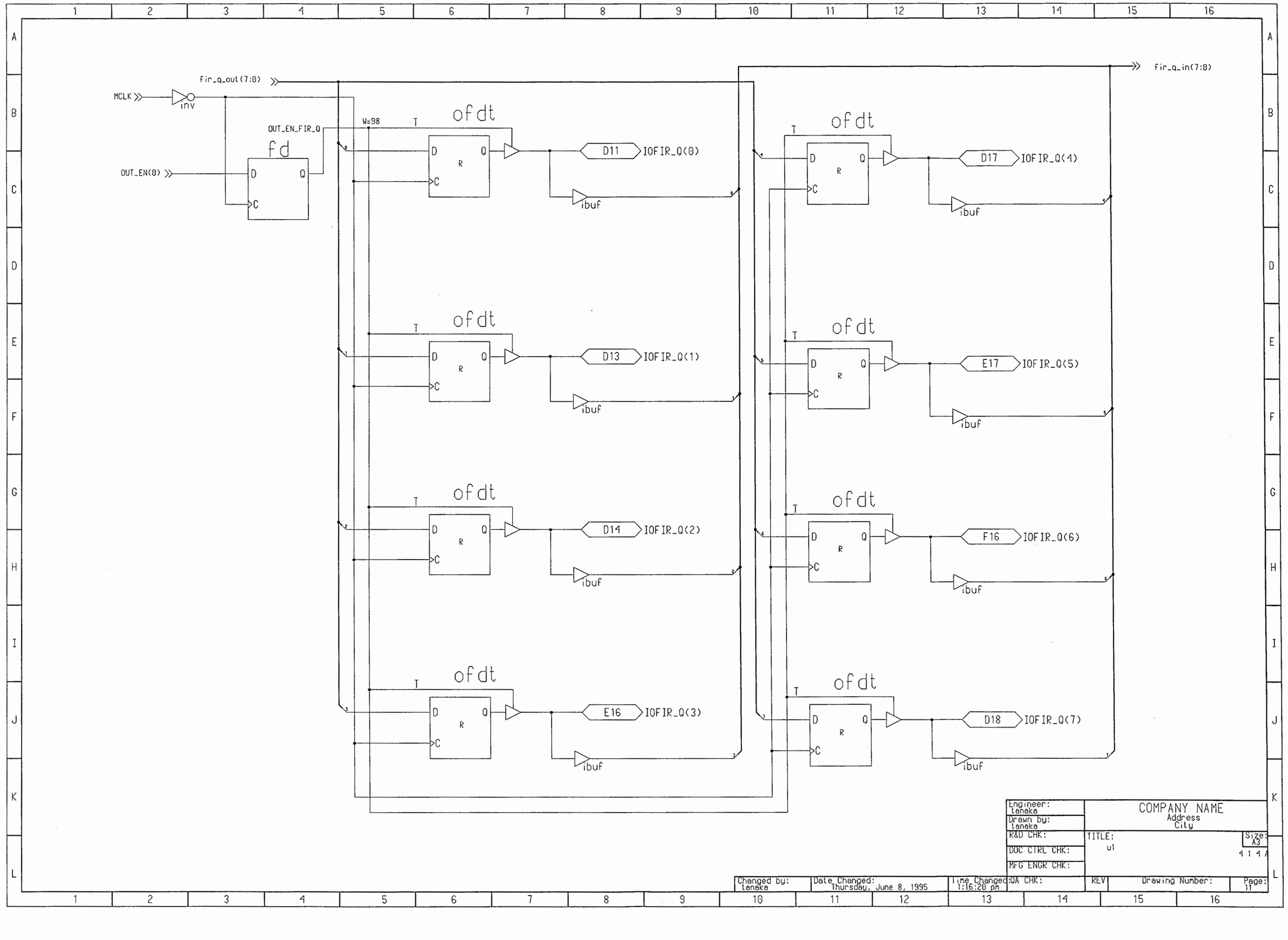


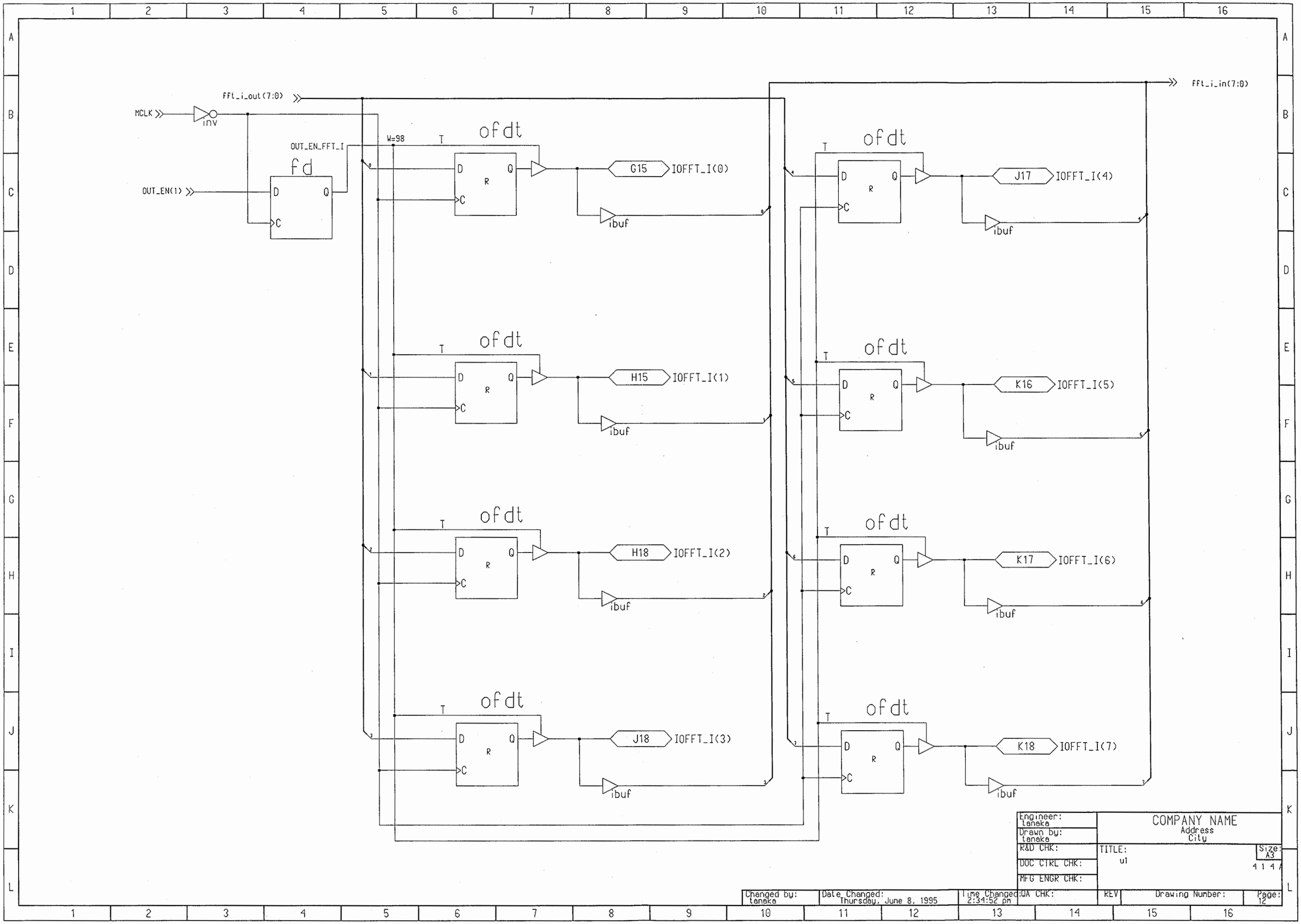
Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: ul
MFG ENGR CHK:	Size: A3
QA CHK:	4 1 4 A

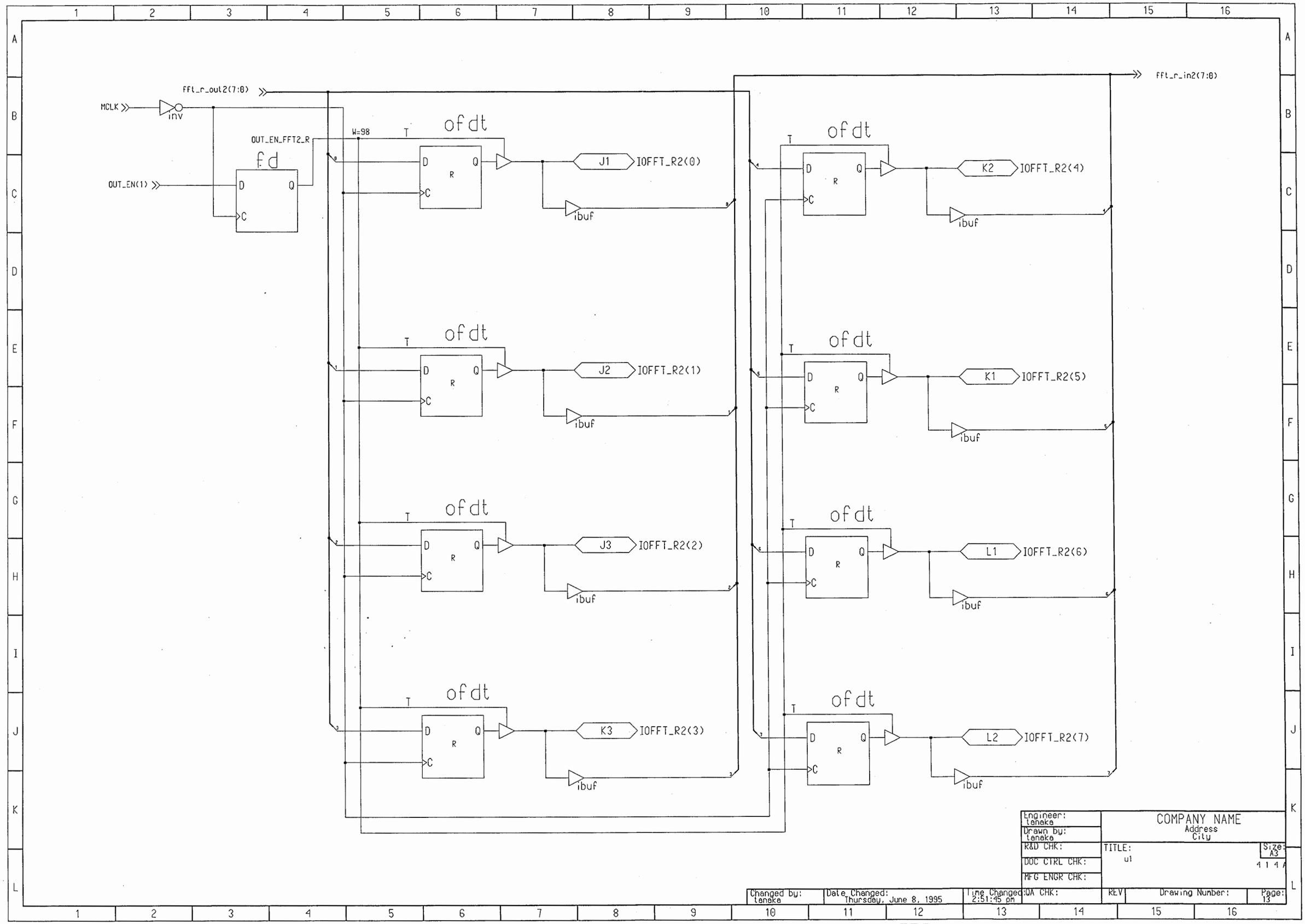
Changed by:	Date Changed:	Time Changed:	REV	Drawing Number:	Page:
tanaka	Thursday, June 8, 1995	12:01:05 pm			9

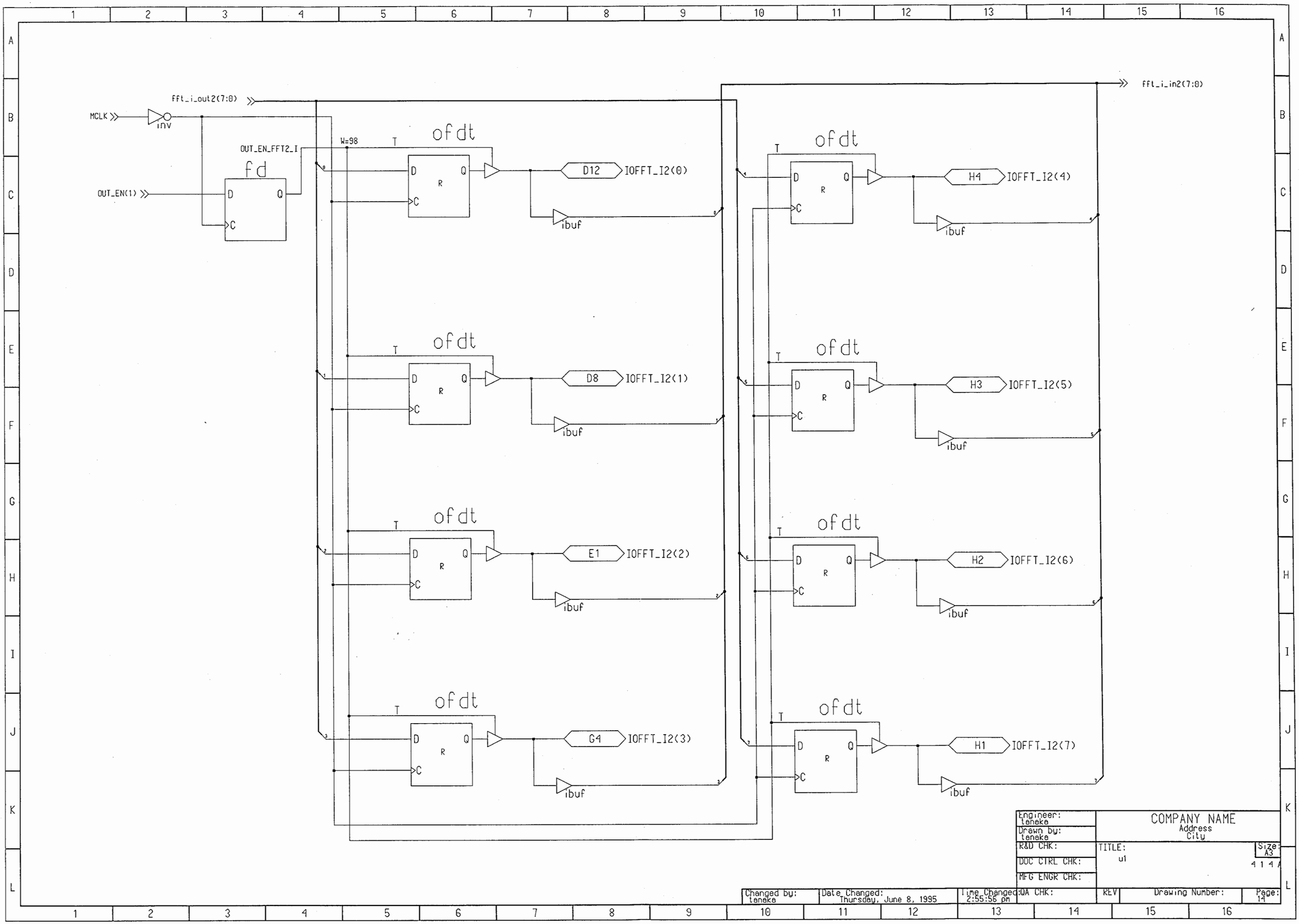


Engineer: teneka	COMPANY NAME Address City				
Drawn by: teneka					
R&D CHK:	TITLE: u1	Size: A3			
DOC CTRL CHK:		4 1 4 A			
MFG ENGR CHK:					
Changed by: teneka	Date Changed: Thursday, June 8, 1995	Time Changed: 1:11:32 pm	QA CHK:	REV	Drawing Number: / Page: 10


fir_q_out(7:0)
MCLK
inv
OUT_EN(0)
fd
OUT_EN_FIR_Q
W=98
ofdt
D
Q
R
C
T
D11
IOFIR_Q(0)
D13
IOFIR_Q(1)
D14
IOFIR_Q(2)
E16
IOFIR_Q(3)
ibuf
fir_q_in(7:0)
D17
IOFIR_Q(4)
E17
IOFIR_Q(5)
F16
IOFIR_Q(6)
D18
IOFIR_Q(7)
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
4 1 4 A
Changed by: taneka
Date Changed: Thursday, June 8, 1995
Time Changed: 1:16:20 pm
QA CHK:
REV
Drawing Number:
Page: 11

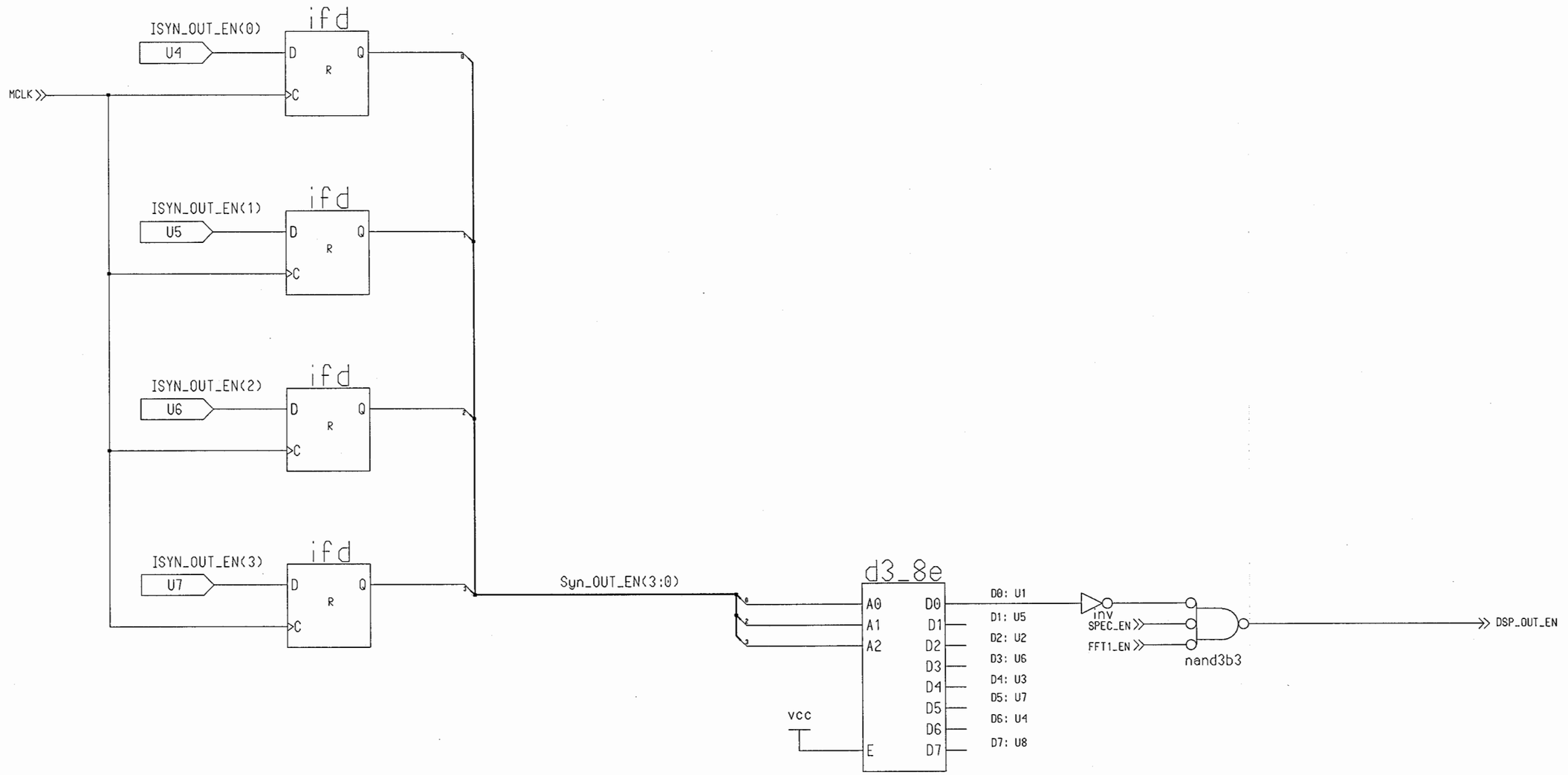

FFT_I_out(7:0)
FFT_I_in(7:0)
MCLK
inv
OUT_EN_FFT_I
OUT_EN(1)
fd
D
Q
C
W=98
T
ofdt
R
G15
IOFFT_I(0)
H15
IOFFT_I(1)
H18
IOFFT_I(2)
J18
IOFFT_I(3)
J17
IOFFT_I(4)
K16
IOFFT_I(5)
K17
IOFFT_I(6)
K18
IOFFT_I(7)
ibuf
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
4 1 4 A
Changed by: tanaka
Date Changed: Thursday, June 8, 1995
Time Changed: 2:34:52 pm
QA CHK:
REV
Drawing Number:
Page: 12


fft_r_out2(7:0)
MCLK
inv
OUT_EN_FFT2_R
fd
OUT_EN(1)
W=98
T
ofdt
J1
IOFFT_R2(0)
ibuf
J2
IOFFT_R2(1)
J3
IOFFT_R2(2)
K3
IOFFT_R2(3)
fft_r_in2(7:0)
K2
IOFFT_R2(4)
K1
IOFFT_R2(5)
L1
IOFFT_R2(6)
L2
IOFFT_R2(7)
D
Q
R
C
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
ut
Size: A3
Changed by: tanaka
Date Changed: Thursday, June 8, 1995
Time Changed: 2:51:45 pm
QA CHK:
REV
Drawing Number:
Page: 13

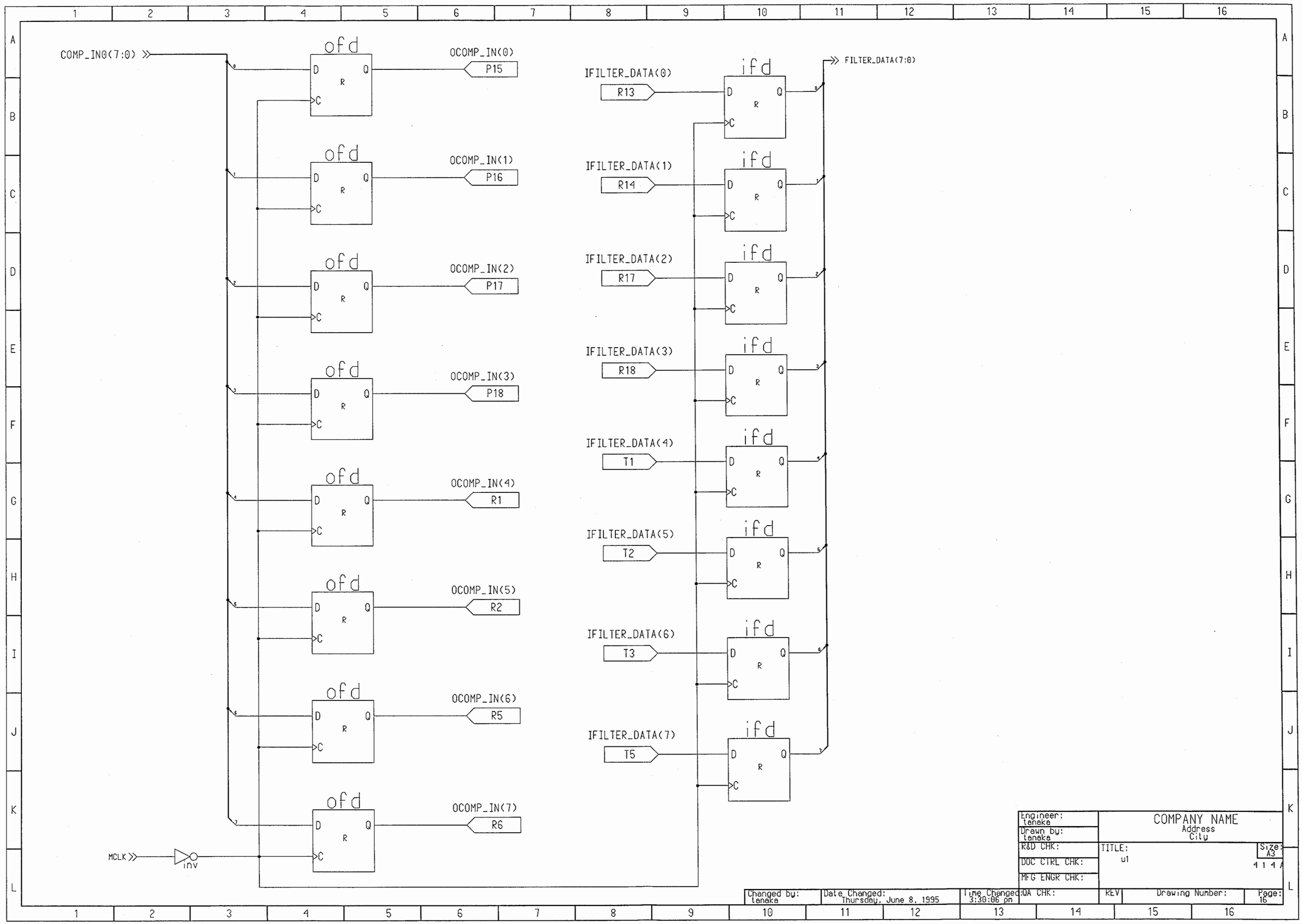


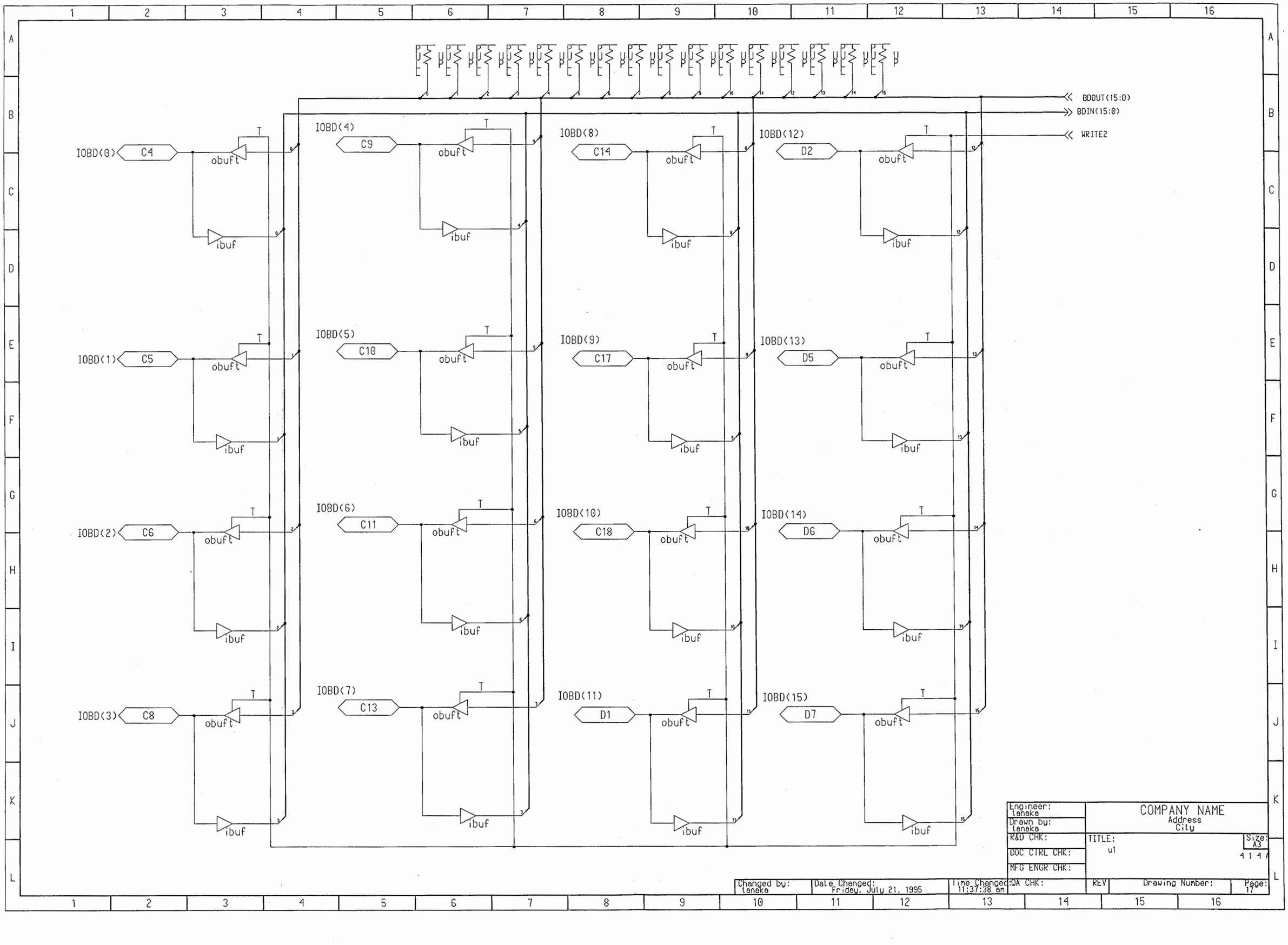
Engineer: teneke	COMPANY NAME
Drawn by: teneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u1
MFG ENGR CHK:	Size: A3
	4 1 4 A

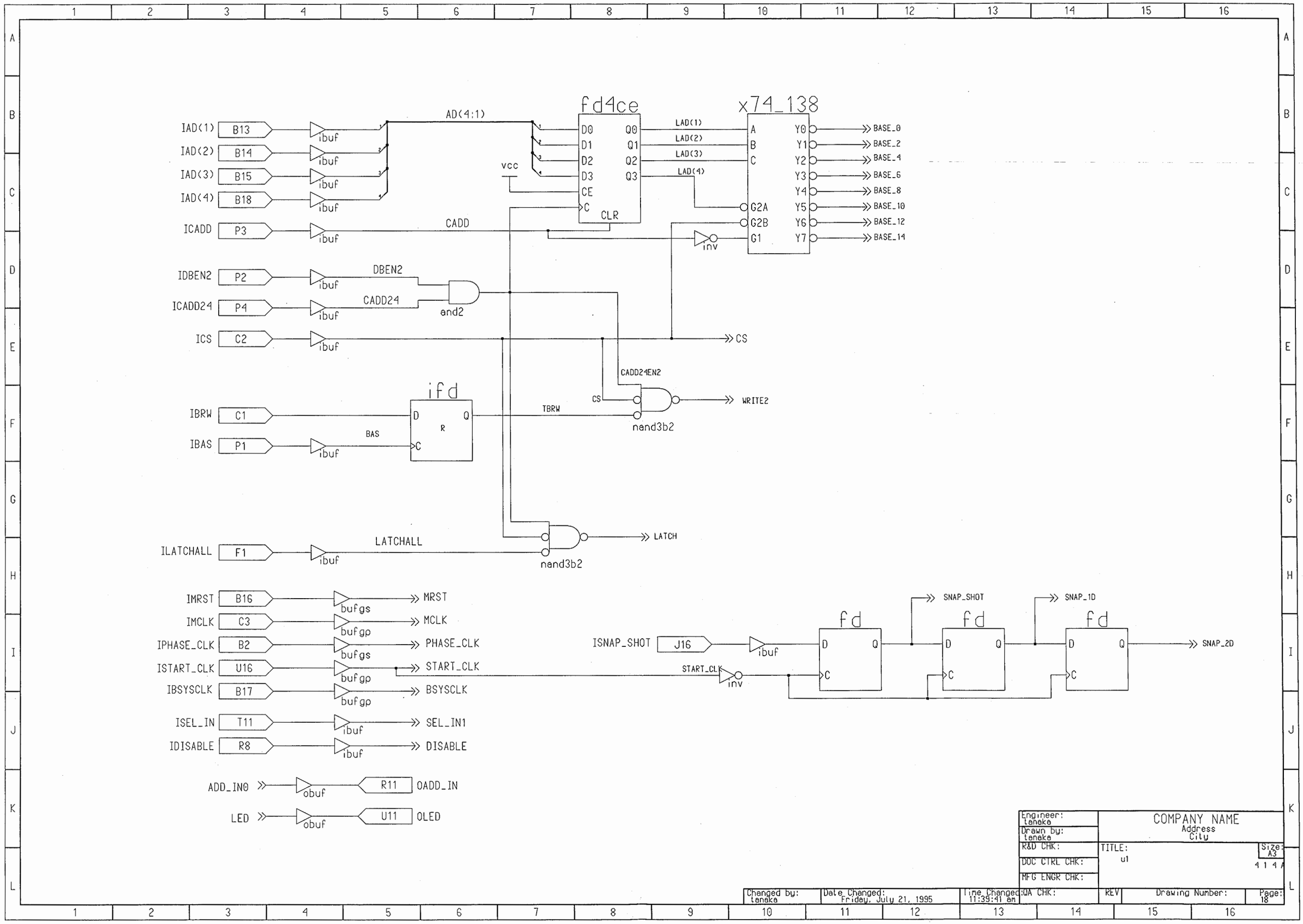
Changed by:	Date Changed:	Time Changed:	QA CHK:	REV	Drawing Number:	Page:
teneke	Thursday, June 8, 1995	2:55:56 pm				14

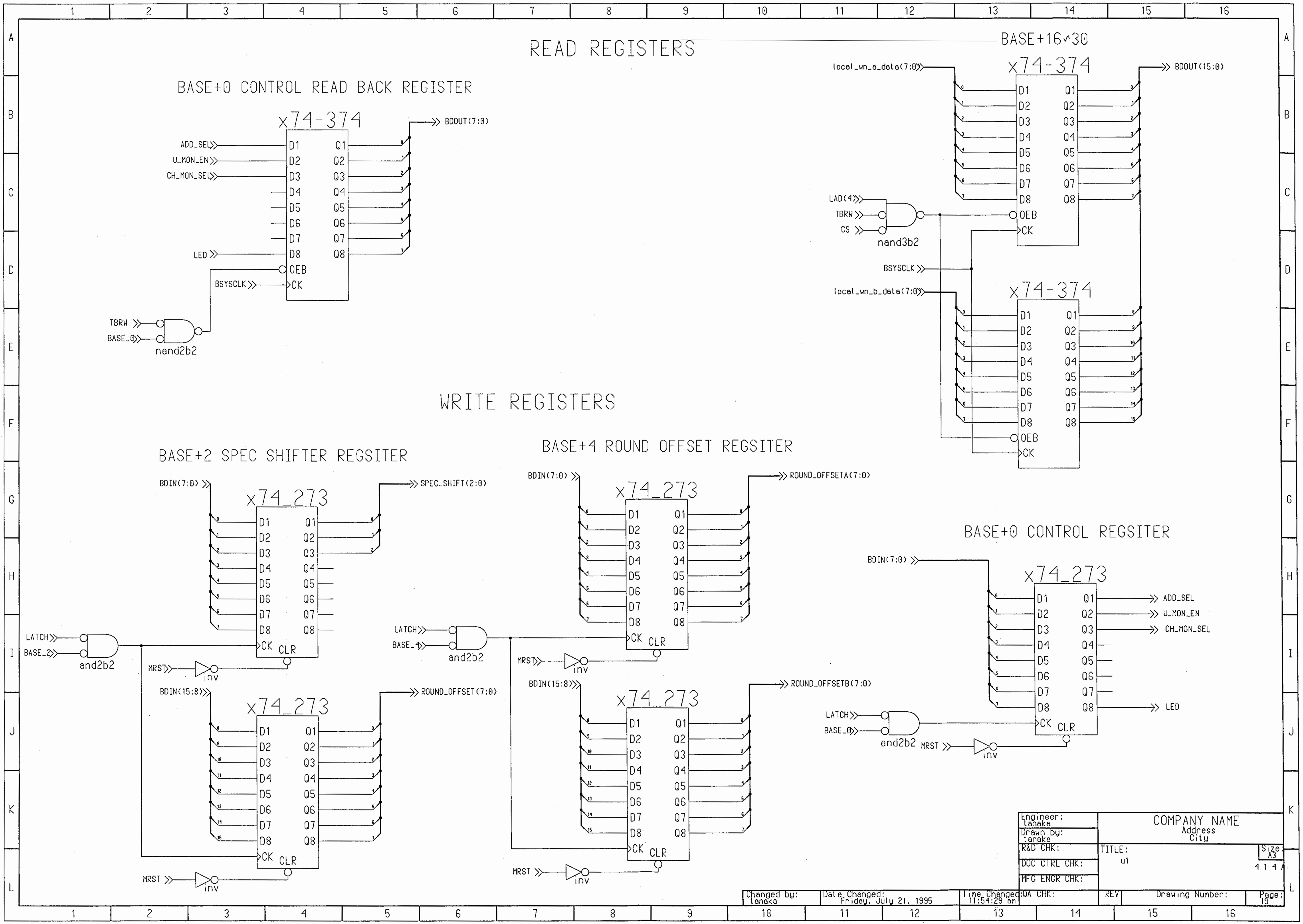


Engineer: laneke	COMPANY NAME
Drawn by: laneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u1
MFG ENGR CHK:	Size: A3
QA CHK:	4 1 4 A
Changed by: laneke	Date Changed: Thursday, June 8, 1995
Time Changed: 2:58:56 pm	REV / Drawing Number: / Page: 15

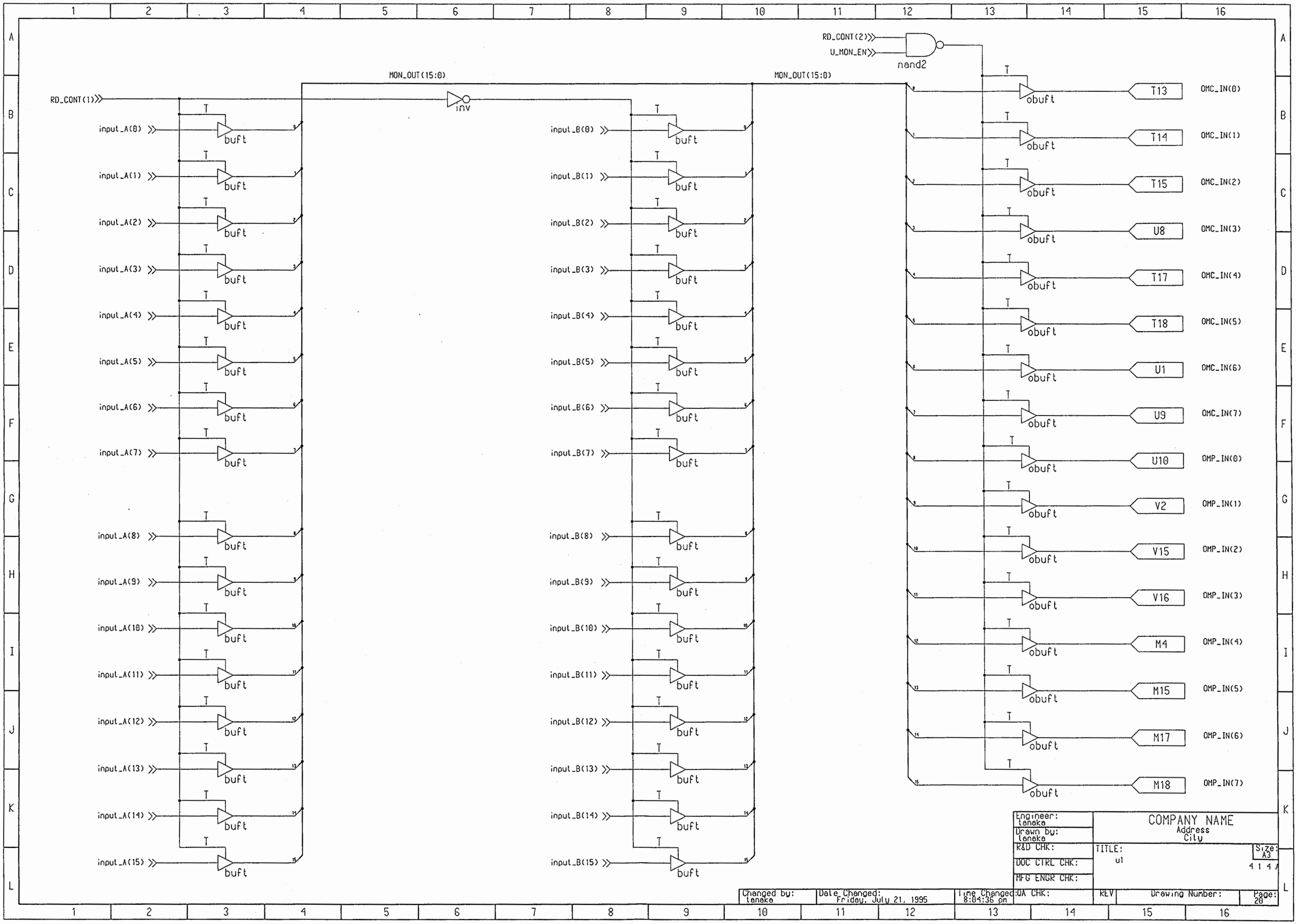

COMP_IN0(7:0)
ofd
D
Q
R
C
OCOMP_IN(0)
P15
OCOMP_IN(1)
P16
OCOMP_IN(2)
P17
OCOMP_IN(3)
P18
OCOMP_IN(4)
R1
OCOMP_IN(5)
R2
OCOMP_IN(6)
R5
OCOMP_IN(7)
R6
MCLK
inv
IFILTER_DATA(0)
R13
IFILTER_DATA(1)
R14
IFILTER_DATA(2)
R17
IFILTER_DATA(3)
R18
IFILTER_DATA(4)
T1
IFILTER_DATA(5)
T2
IFILTER_DATA(6)
T3
IFILTER_DATA(7)
T5
ifd
FILTER_DATA(7:0)
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
QA CHK:
REV
Drawing Number:
Page: 16
Changed by: tanaka
Date Changed: Thursday, June 8, 1995
Time Changed: 3:30:06 pm


BDOUT(15:0)
BDIN(15:0)
WRITE2
IOBD(0)
C4
IOBD(4)
C9
IOBD(8)
C14
IOBD(12)
D2
IOBD(1)
C5
IOBD(5)
C10
IOBD(9)
C17
IOBD(13)
D5
IOBD(2)
C6
IOBD(6)
C11
IOBD(10)
C18
IOBD(14)
D6
IOBD(3)
C8
IOBD(7)
C13
IOBD(11)
D1
IOBD(15)
D7
T
obuft
ibuf
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
Changed by: tanaka
Date Changed: Friday, July 21, 1995
Time Changed: 11:37:38 am
QA CHK:
REV
Drawing Number:
Page: 17

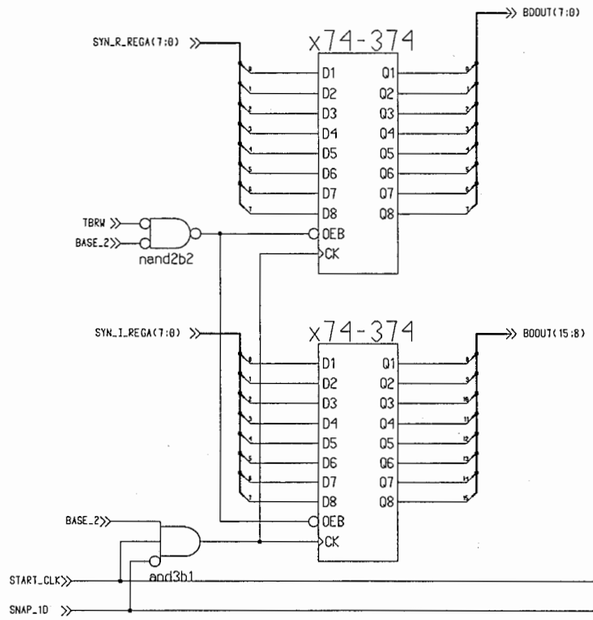

IAD(1)
B13
IAD(2)
B14
IAD(3)
B15
IAD(4)
B18
ICADD
P3
ibuf
AD(4:1)
CADD
fd4ce
D0
D1
D2
D3
CE
C
CLR
Q0
Q1
Q2
Q3
vcc
LAD(1)
LAD(2)
LAD(3)
LAD(4)
x74_138
A
B
C
G2A
G2B
G1
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
BASE_0
BASE_2
BASE_4
BASE_6
BASE_8
BASE_10
BASE_12
BASE_14
inv
IDBEN2
P2
ICADD24
P4
DBEN2
CADD24
and2
ICS
C2
CS
IBRW
C1
IBAS
P1
BAS
ifd
D
Q
R
C
TBRW
CADD24EN2
nand3b2
WRITE2
ILATCHALL
F1
LATCHALL
nand3b2
LATCH
IMRST
B16
IMCLK
C3
IPHASE_CLK
B2
ISTART_CLK
U16
IBSYSCLK
B17
bufgs
bufgp
MRST
MCLK
PHASE_CLK
START_CLK
BSYSCLK
ISEL_IN
T11
IDISABLE
R8
SEL_IN1
DISABLE
ADD_IN0
R11
OADD_IN
LED
U11
OLED
obuf
ISNAP_SHOT
J16
START_CLK
fd
SNAP_SHOT
SNAP_1D
SNAP_2D
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE:
u1
Size: A3
4 1 4 A
Changed by: tanaka
Date Changed: Friday, July 21, 1995
Time Changed: 11:39:41 am
REV
Drawing Number:
Page: 18



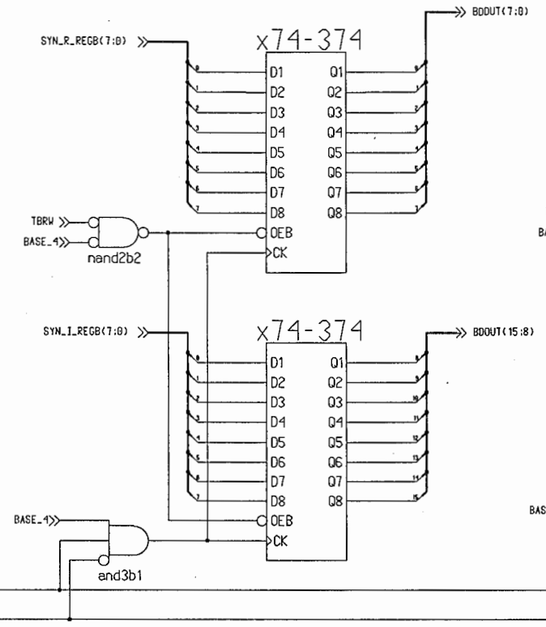
READ REGISTERS
BASE+0 CONTROL READ BACK REGISTER
x74-374
ADD_SEL
U_MON_EN
CH_MON_SEL
LED
BSYSCLK
BDOUT(7:0)
TBRW
BASE_0
nand2b2
BASE+16~30
x74-374
local_wn_a_data(7:0)
BDOUT(15:0)
LAD(4)
TBRW
CS
nand3b2
BSYSCLK
local_wn_b_data(7:0)
x74-374
WRITE REGISTERS
BASE+2 SPEC SHIFTER REGSITER
BDIN(7:0)
x74_273
SPEC_SHIFT(2:0)
LATCH
BASE_2
and2b2
MRST
inv
BDIN(15:8)
x74_273
ROUND_OFFSET(7:0)
MRST
inv
BASE+4 ROUND OFFSET REGSITER
BDIN(7:0)
x74_273
ROUND_OFFSETA(7:0)
LATCH
BASE_4
and2b2
MRST
inv
BDIN(15:8)
x74_273
ROUND_OFFSETB(7:0)
MRST
inv
BASE+0 CONTROL REGSITER
BDIN(7:0)
x74_273
ADD_SEL
U_MON_EN
CH_MON_SEL
LED
LATCH
BASE_0
and2b2
MRST
inv
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE: u1
Size: A3
REV
Drawing Number:
Page: 19
Changed by: tanaka
Date Changed: Friday, July 21, 1995
Time Changed: 11:54:29 am


RD_CONT(1)
RD_CONT(2)
U_MON_EN
nand2
MON_OUT(15:0)
MON_OUT(15:0)
inv
input_A(0)
input_A(1)
input_A(2)
input_A(3)
input_A(4)
input_A(5)
input_A(6)
input_A(7)
input_A(8)
input_A(9)
input_A(10)
input_A(11)
input_A(12)
input_A(13)
input_A(14)
input_A(15)
input_B(0)
input_B(1)
input_B(2)
input_B(3)
input_B(4)
input_B(5)
input_B(6)
input_B(7)
input_B(8)
input_B(9)
input_B(10)
input_B(11)
input_B(12)
input_B(13)
input_B(14)
input_B(15)
buft
obuft
T13
T14
T15
U8
T17
T18
U1
U9
U10
V2
V15
V16
M4
M15
M17
M18
OMC_IN(0)
OMC_IN(1)
OMC_IN(2)
OMC_IN(3)
OMC_IN(4)
OMC_IN(5)
OMC_IN(6)
OMC_IN(7)
OMP_IN(0)
OMP_IN(1)
OMP_IN(2)
OMP_IN(3)
OMP_IN(4)
OMP_IN(5)
OMP_IN(6)
OMP_IN(7)
Engineer: taneke
Drawn by: taneke
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE: u1
Size: A3
Changed by: taneke
Date Changed: Friday, July 21, 1995
Time Changed: 8:04:36 pm
QA CHK:
REV
Drawing Number:
Page: 20

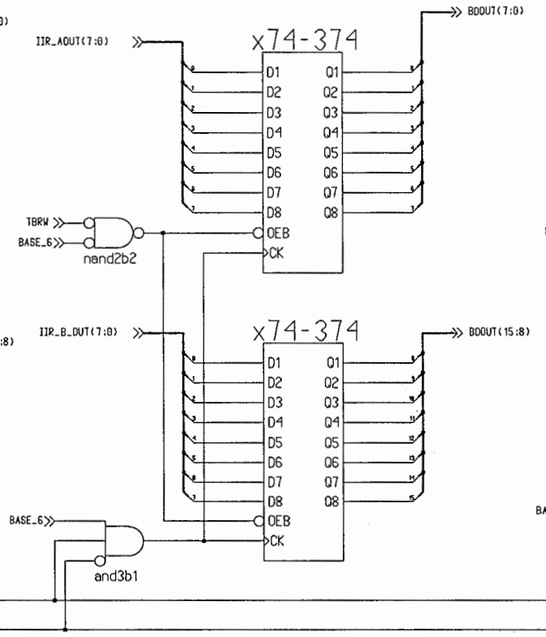
BASE +2 Ach SYN DATA REGISTER



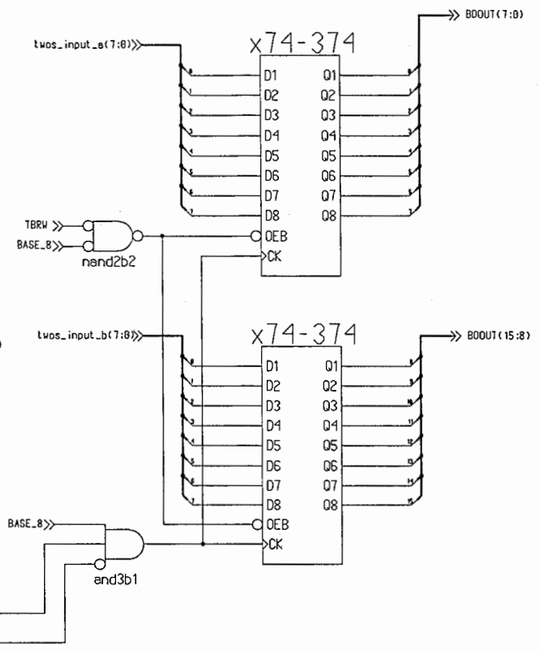
BASE +4 Bch SYN DATA REGISTER



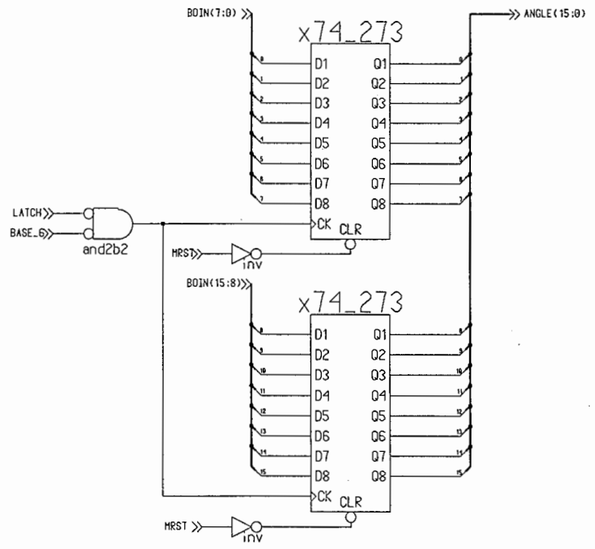
BASE +6 SPECTRUM DATA REGISTER



BASE +8 INPUT DATA REGISTER



BASE+6 ANGLE REGSITER

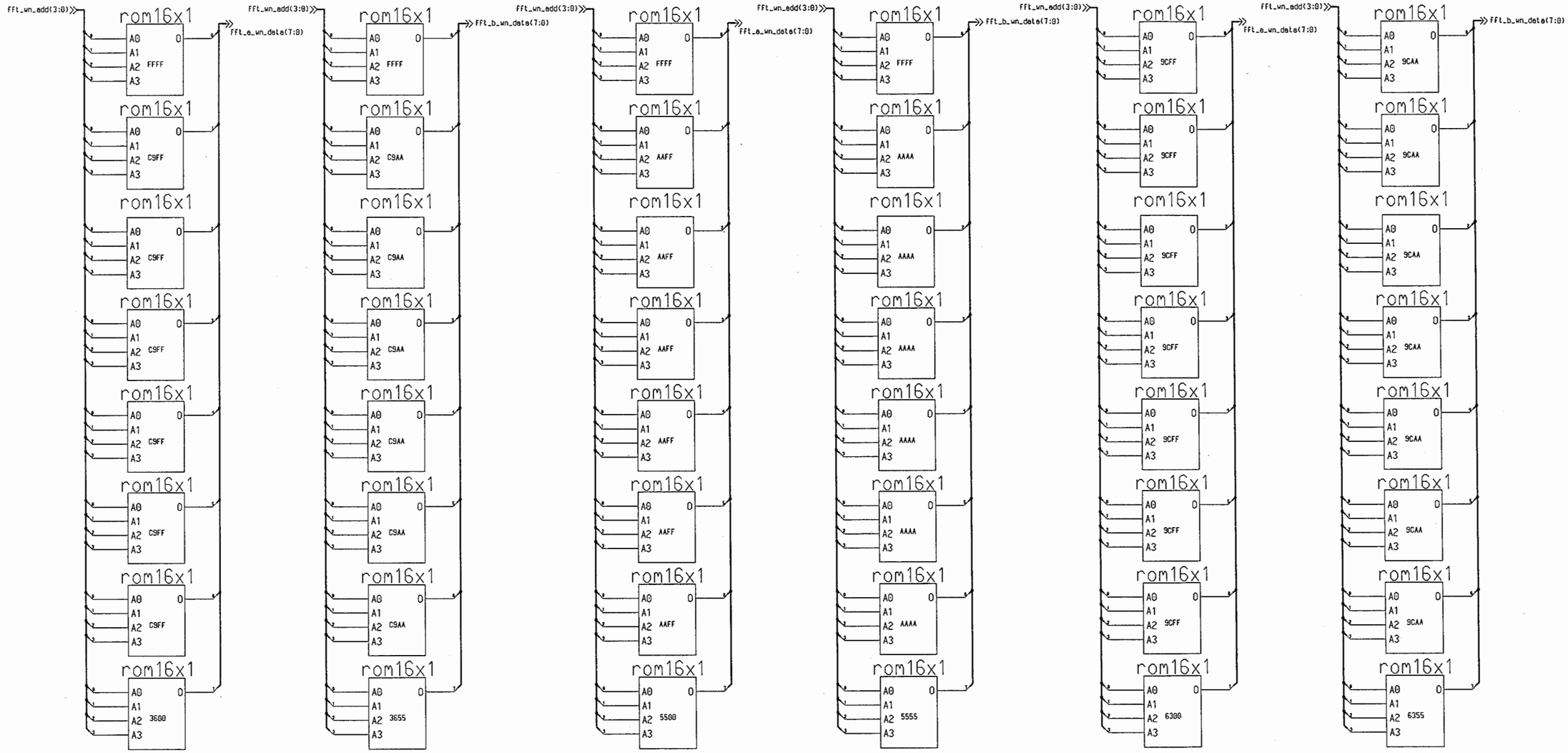


Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u1
MFG ENGR CHK:	Size: A2
Changed by: tanaka	Date Changed: Tuesday, September 12, 1995
Time Changed: 10:43:49 am	QA CHK:
REV	Drawing Number:
Page: 21	

U2 FFT ROM

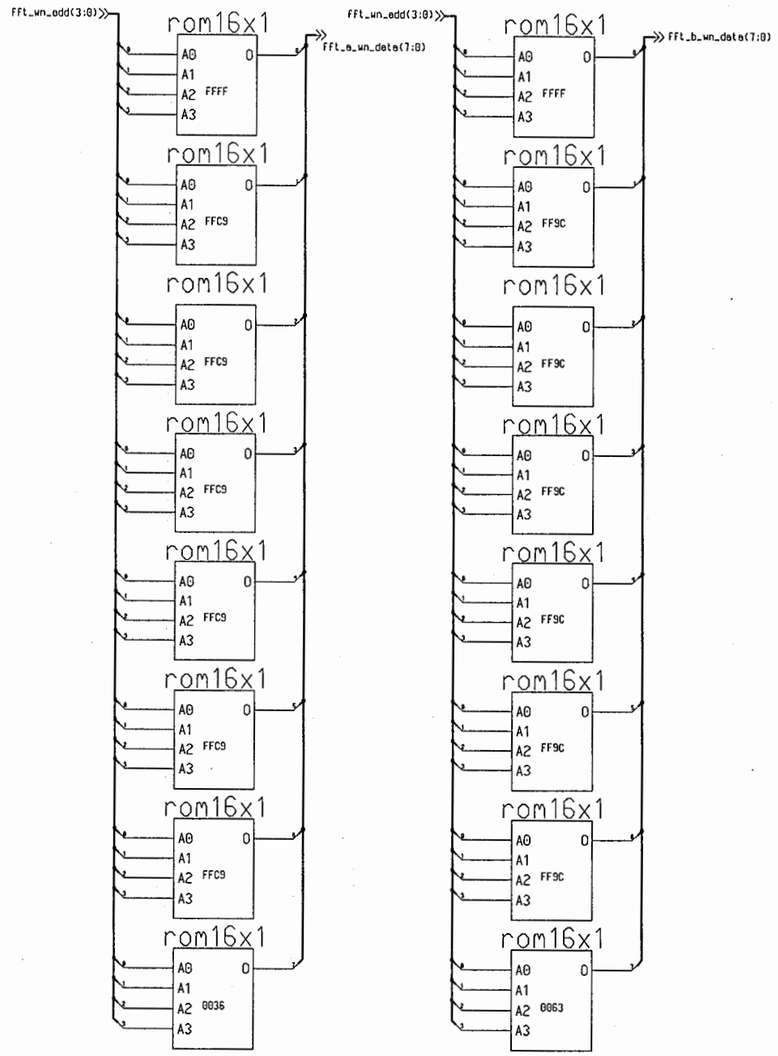
U3 FFT ROM

U4 FFT ROM

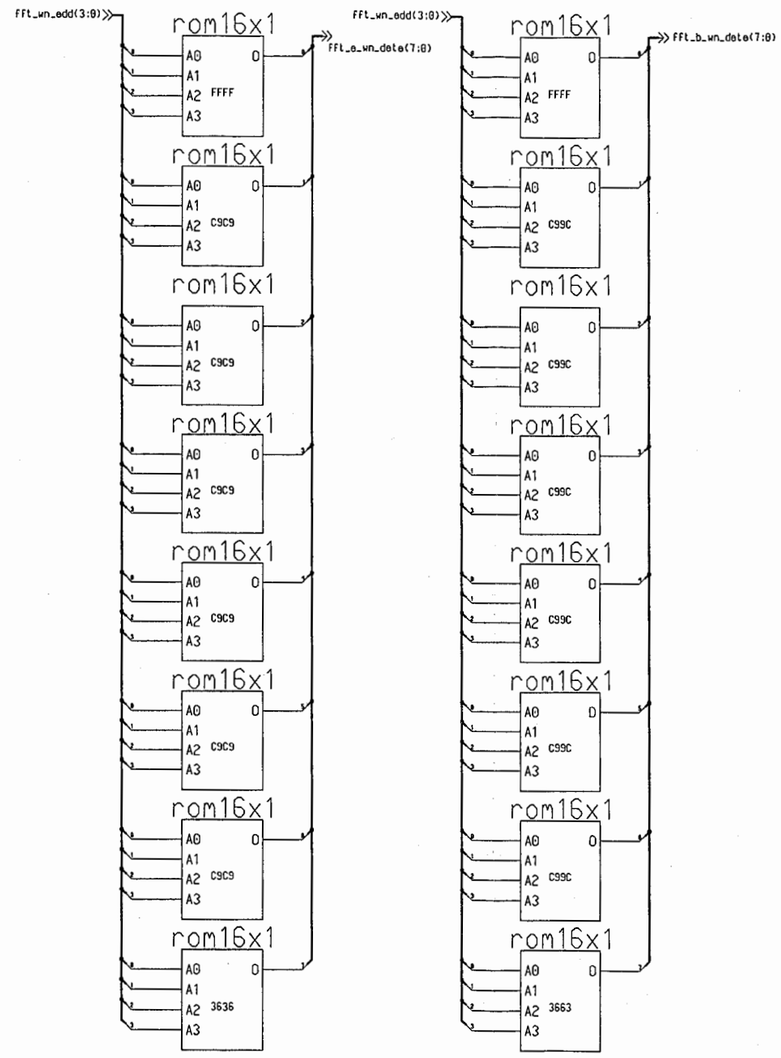


Engineer: teneka	COMPANY NAME
Drawn By: teneka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: FFT_ROMS
MFG ENGR CHK:	Size: A2
Changed By: teneka	Date Changed: Friday, July 21, 1995
Time Changed: 4:03:40 pm	QA CHK:
REV	Drawing Number:
Page: 1	

U5 FFT ROM



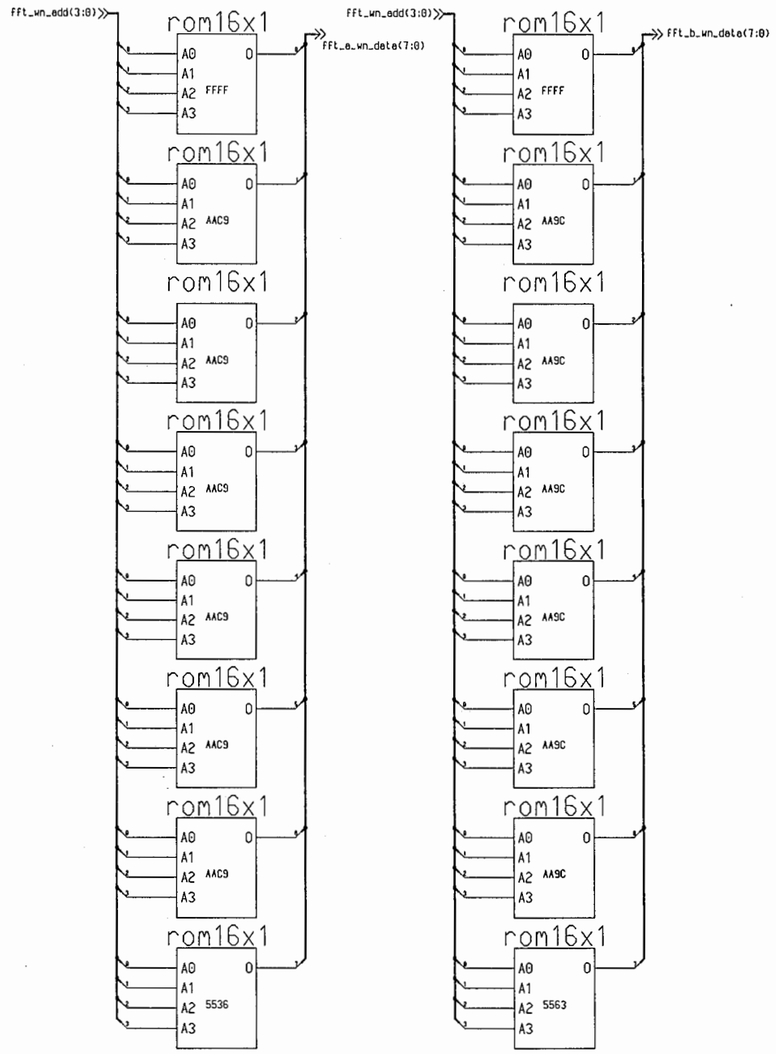
U6 FFT ROM



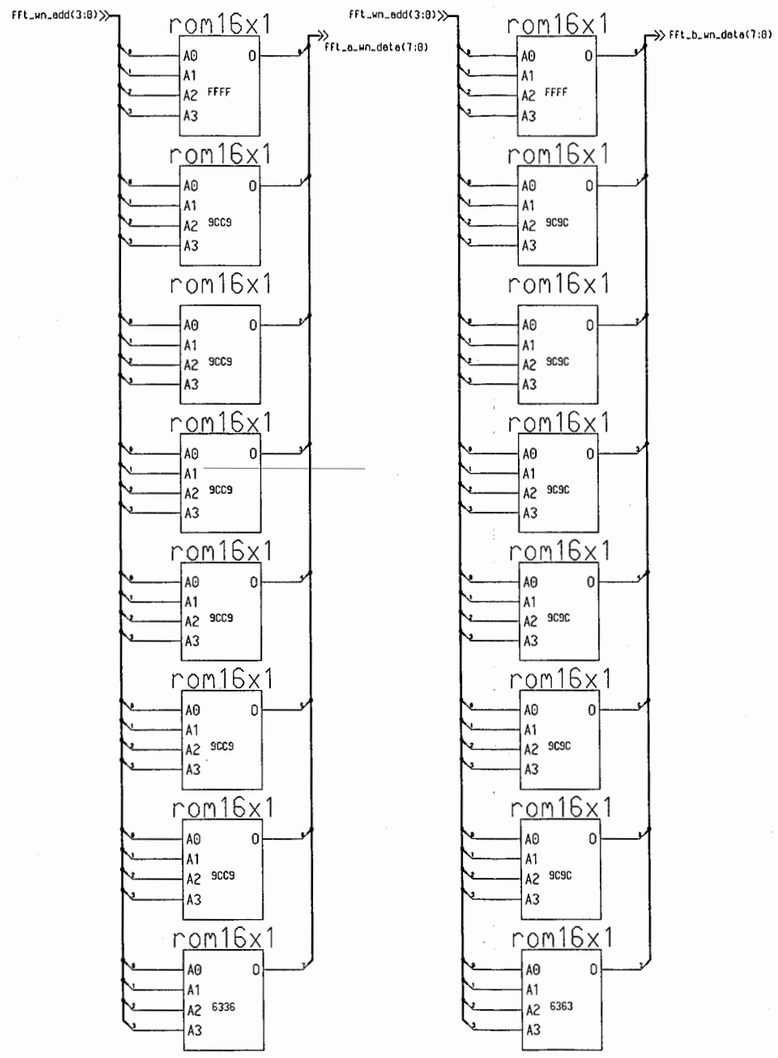
Engineer: taneke	COMPANY NAME
Drawn by: taneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: FFT_ROMS
MFG ENGR CHK:	Size: A2
	4 1 4 7

Changed by: taneke	Date Changed: Friday, July 21, 1995	Time Changed: 1:04:48 pm	QA CHK:	REV	Drawing Number:	Page: 2

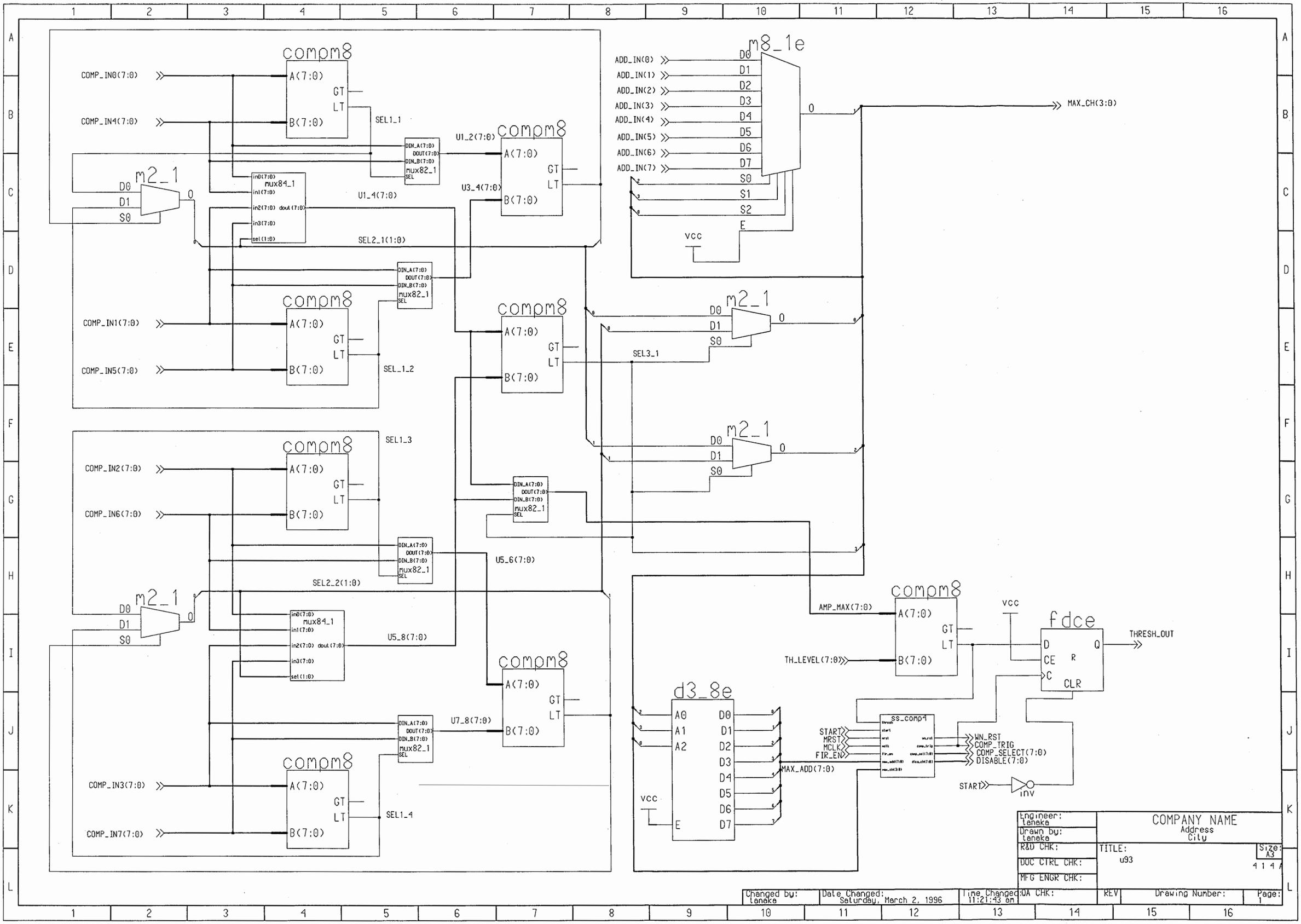
U7 FFT ROM

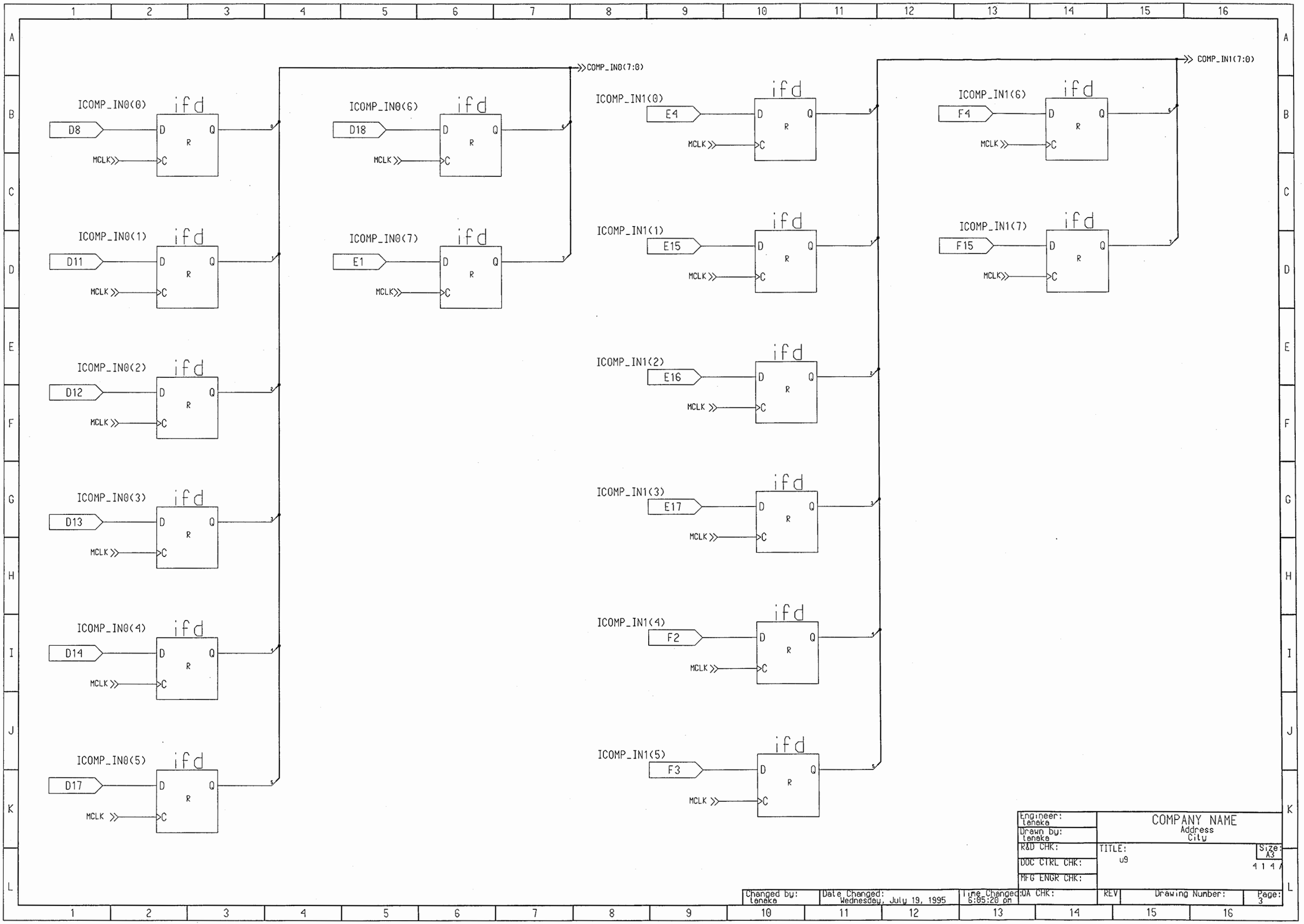


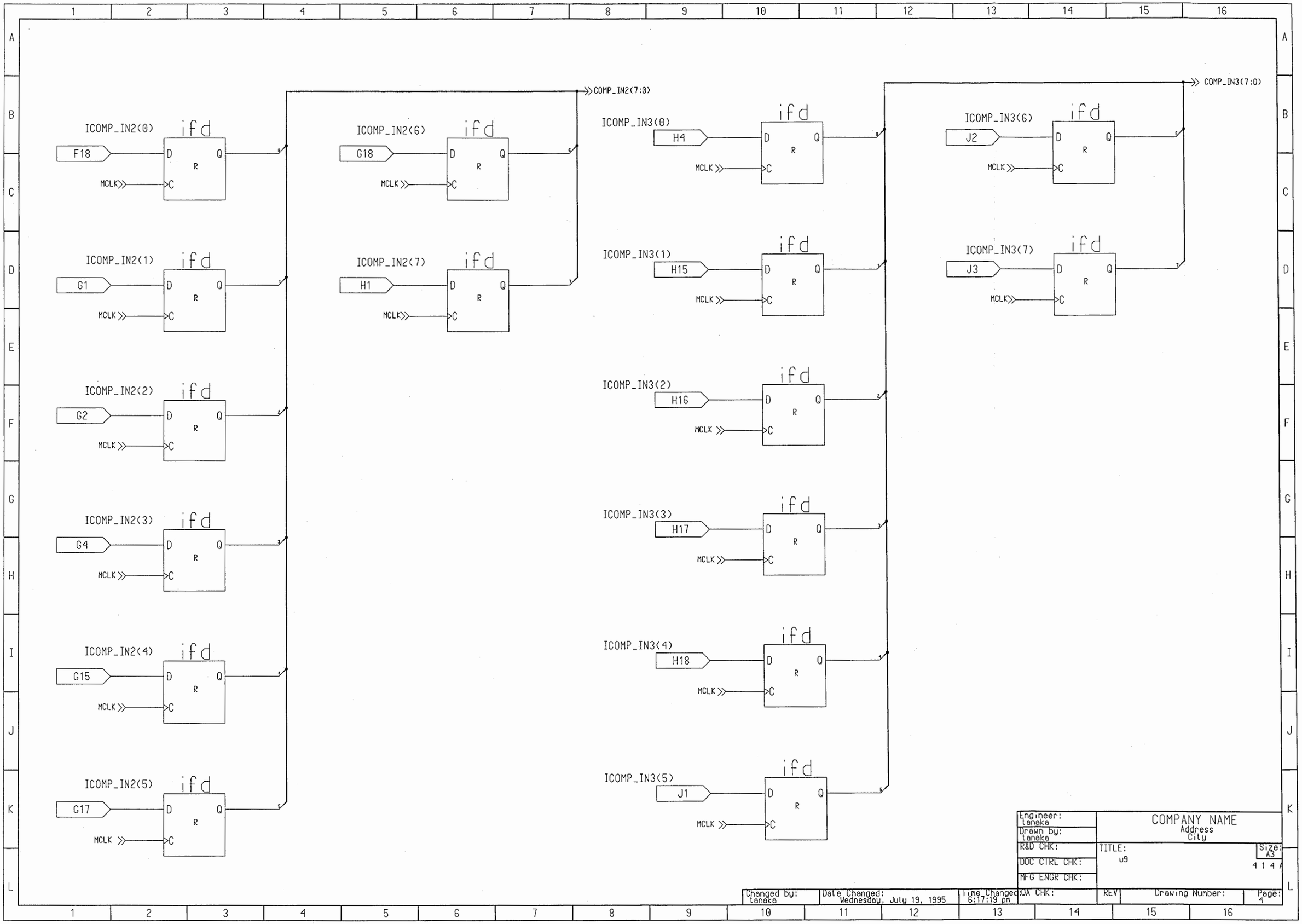
U8 FFT ROM

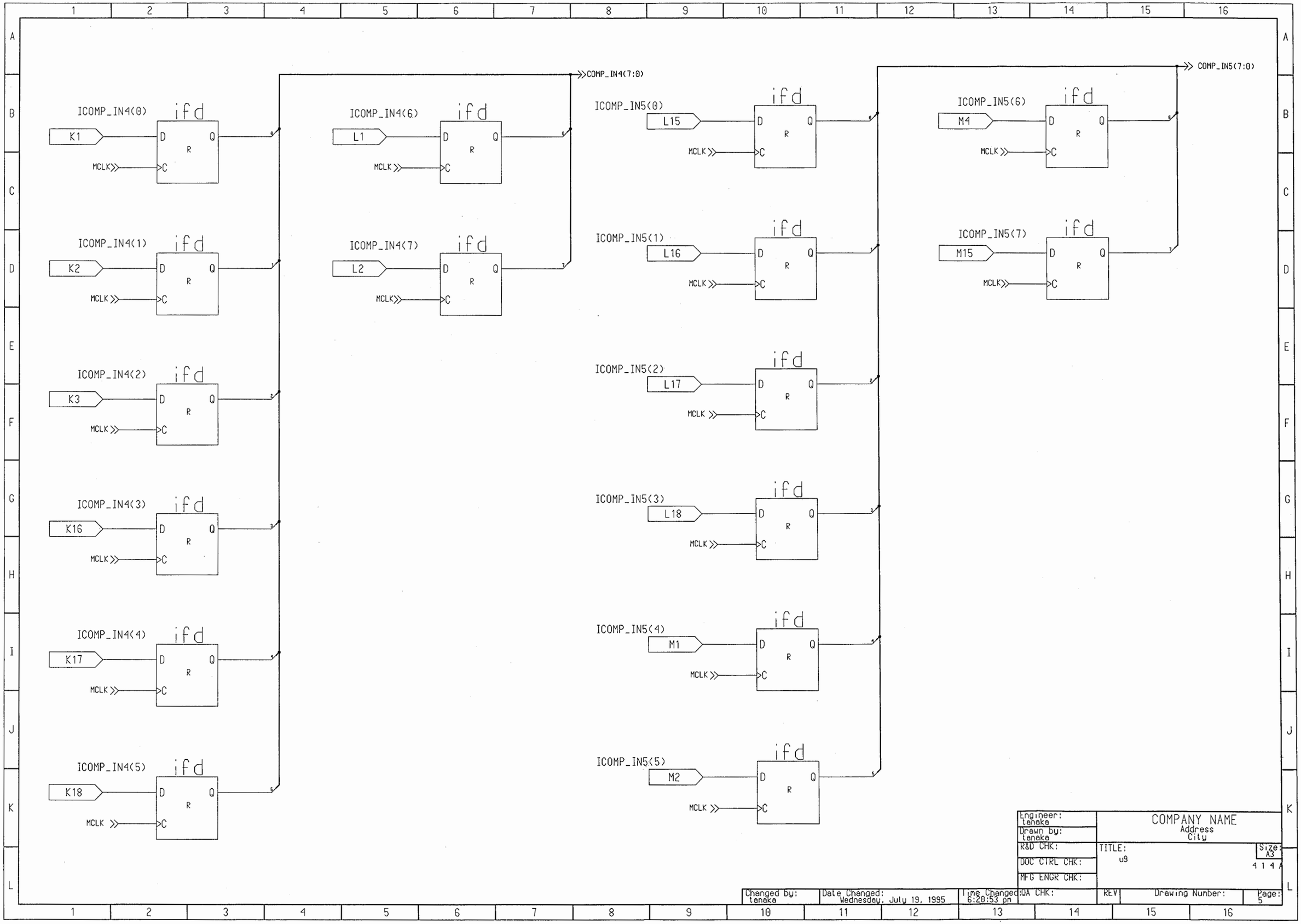


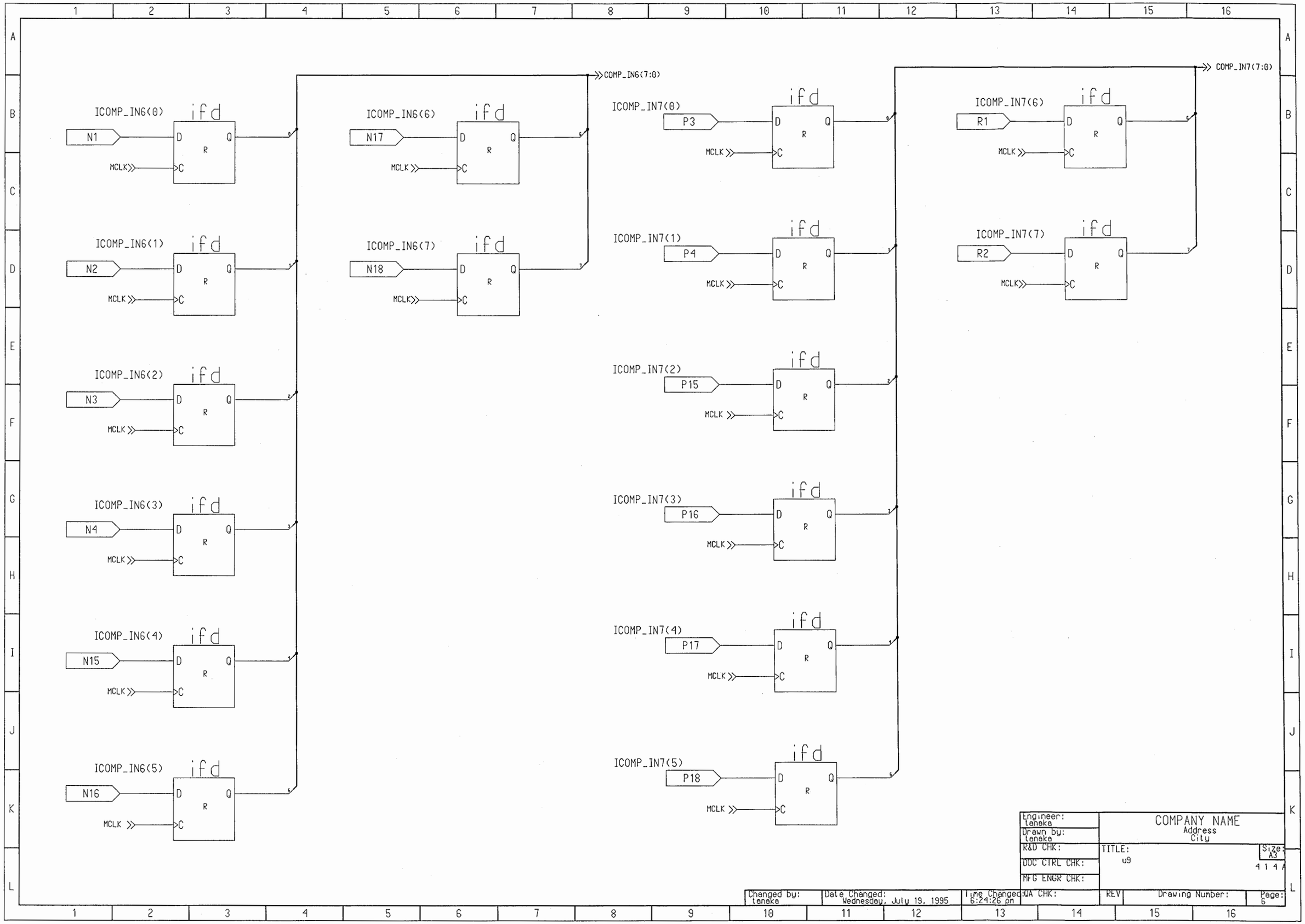
Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: FFT_ROMS
MFG ENGR CHK:	Size: A
QA CHK:	REV: 1.1.1.1
Changed by: tanaka	Drawing Number:
Date Changed: Friday, July 21, 1995	Page: 3
Time Changed: 4:05:22 pm	

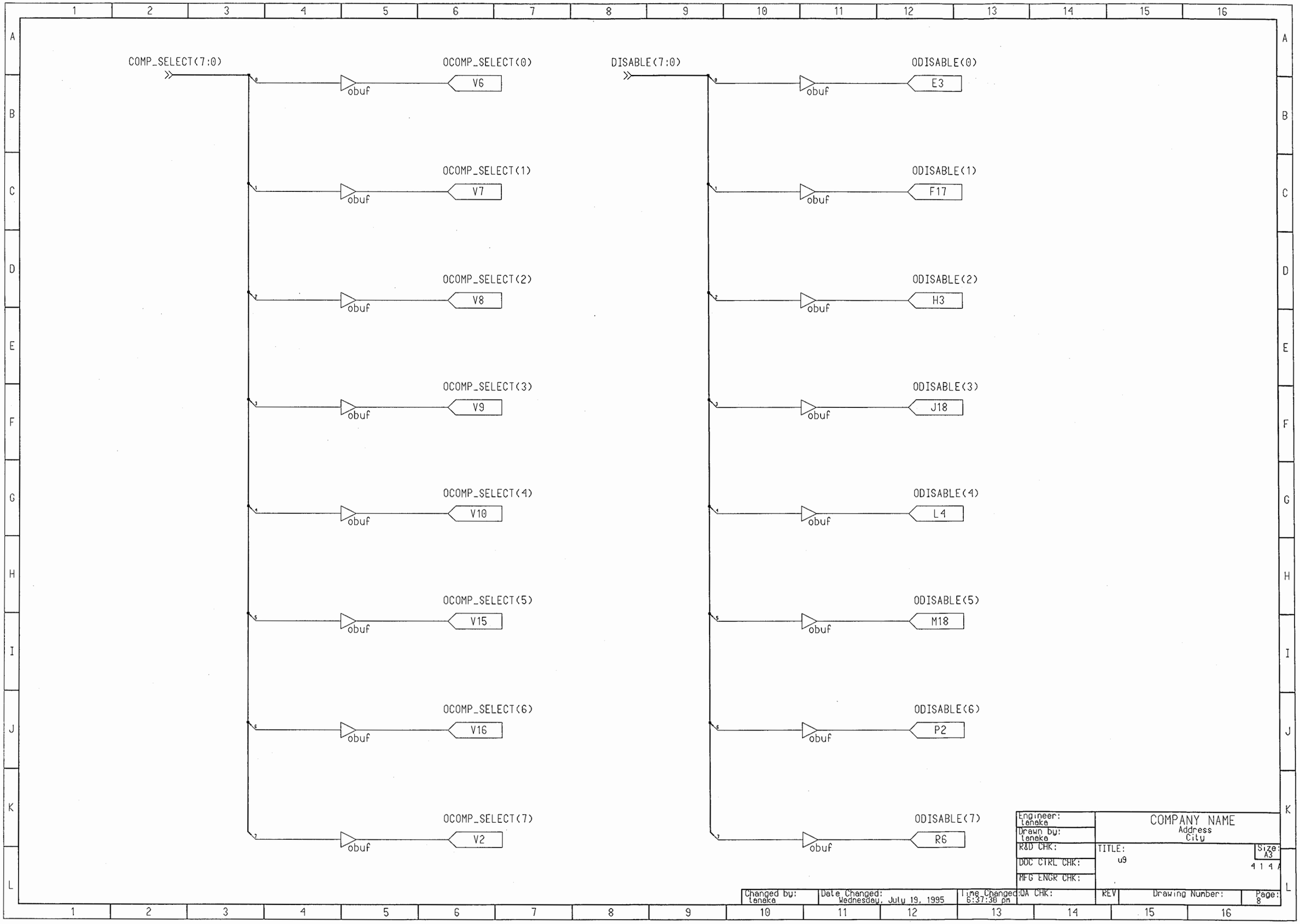

compm8
A(7:0)
B(7:0)
GT
LT
COMP_IN0(7:0)
COMP_IN4(7:0)
SEL1_1
m2_1
D0
D1
S0
O
mux84_1
in0(7:0)
in1(7:0)
in2(7:0)
in3(7:0)
sel(1:0)
dout(7:0)
U1_4(7:0)
mux82_1
DIN_A(7:0)
DIN_B(7:0)
DOUT(7:0)
SEL
U1_2(7:0)
U3_4(7:0)
SEL2_1(1:0)
COMP_IN1(7:0)
COMP_IN5(7:0)
SEL_1_2
SEL1_3
COMP_IN2(7:0)
COMP_IN6(7:0)
U5_6(7:0)
SEL2_2(1:0)
U5_8(7:0)
U7_8(7:0)
COMP_IN3(7:0)
COMP_IN7(7:0)
SEL1_4
m8_1e
ADD_IN(0)
ADD_IN(1)
ADD_IN(2)
ADD_IN(3)
ADD_IN(4)
ADD_IN(5)
ADD_IN(6)
ADD_IN(7)
D0
D1
D2
D3
D4
D5
D6
D7
S0
S1
S2
E
VCC
MAX_CH(3:0)
SEL3_1
AMP_MAX(7:0)
TH_LEVEL(7:0)
fdce
D
Q
CE
R
C
CLR
THRESH_OUT
d3_8e
A0
A1
A2
E
MAX_ADD(7:0)
ss_comp4
START
MRST
MCLK
FIR_EN
WN_RST
COMP_TRIG
COMP_SELECT(7:0)
DISABLE(7:0)
inv
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u93
Size: A3
Changed by: taneka
Date Changed: Saturday, March 2, 1996
Time Changed: 11:21:43 am
QA CHK:
REV
Drawing Number:
Page: 1

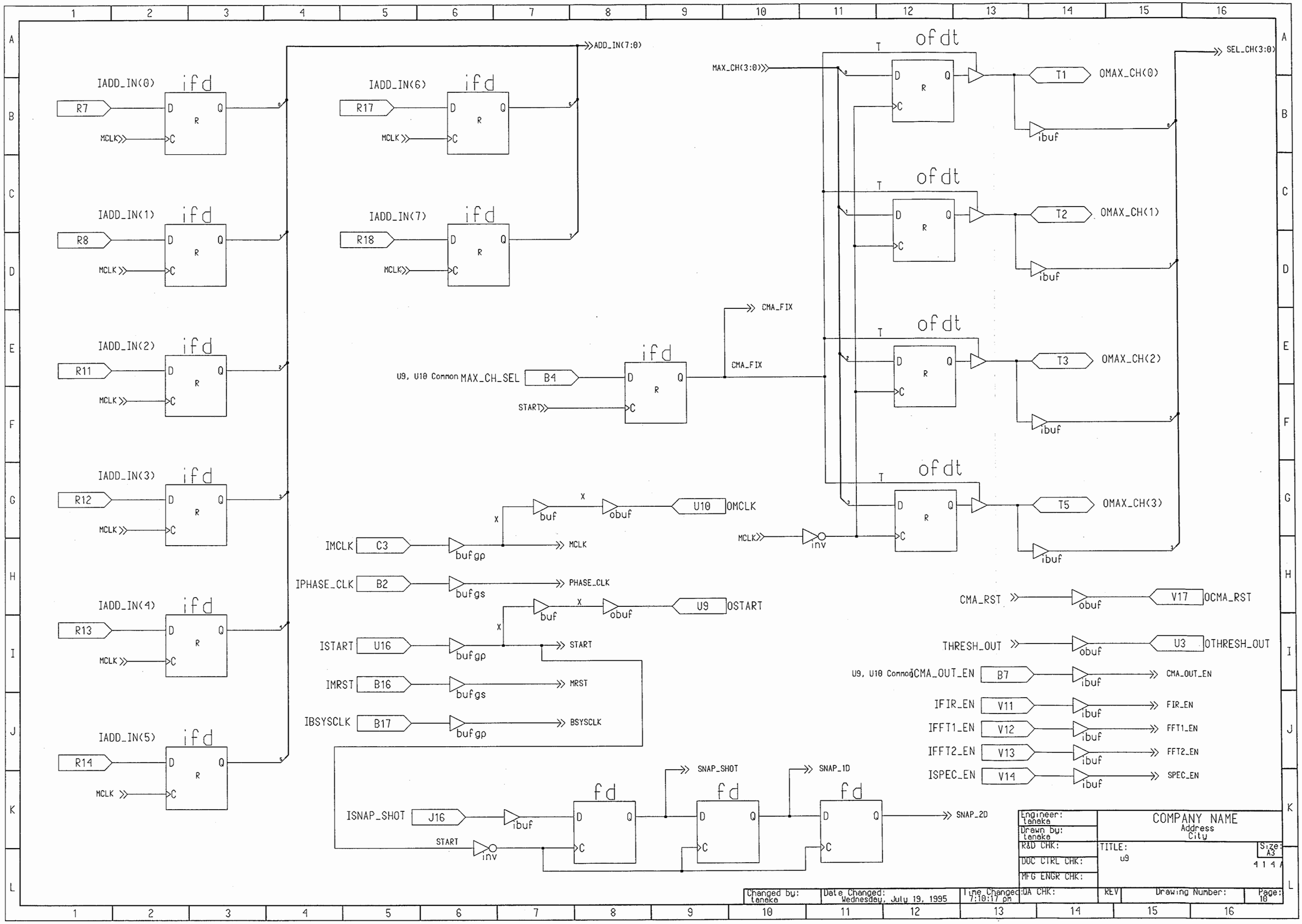

ICOMP_IN0(0)
D8
MCLK
ICOMP_IN0(1)
D11
MCLK
ICOMP_IN0(2)
D12
MCLK
ICOMP_IN0(3)
D13
MCLK
ICOMP_IN0(4)
D14
MCLK
ICOMP_IN0(5)
D17
MCLK
ICOMP_IN0(6)
D18
MCLK
ICOMP_IN0(7)
E1
MCLK
COMP_IN0(7:0)
ICOMP_IN1(0)
E4
MCLK
ICOMP_IN1(1)
E15
MCLK
ICOMP_IN1(2)
E16
MCLK
ICOMP_IN1(3)
E17
MCLK
ICOMP_IN1(4)
F2
MCLK
ICOMP_IN1(5)
F3
MCLK
ICOMP_IN1(6)
F4
MCLK
ICOMP_IN1(7)
F15
MCLK
COMP_IN1(7:0)
ifd
D
Q
R
C
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
4 1 4 A
Changed by: taneka
Date Changed: Wednesday, July 19, 1995
Time Changed: 6:05:20 pm
QA CHK:
REV
Drawing Number:
Page: 3

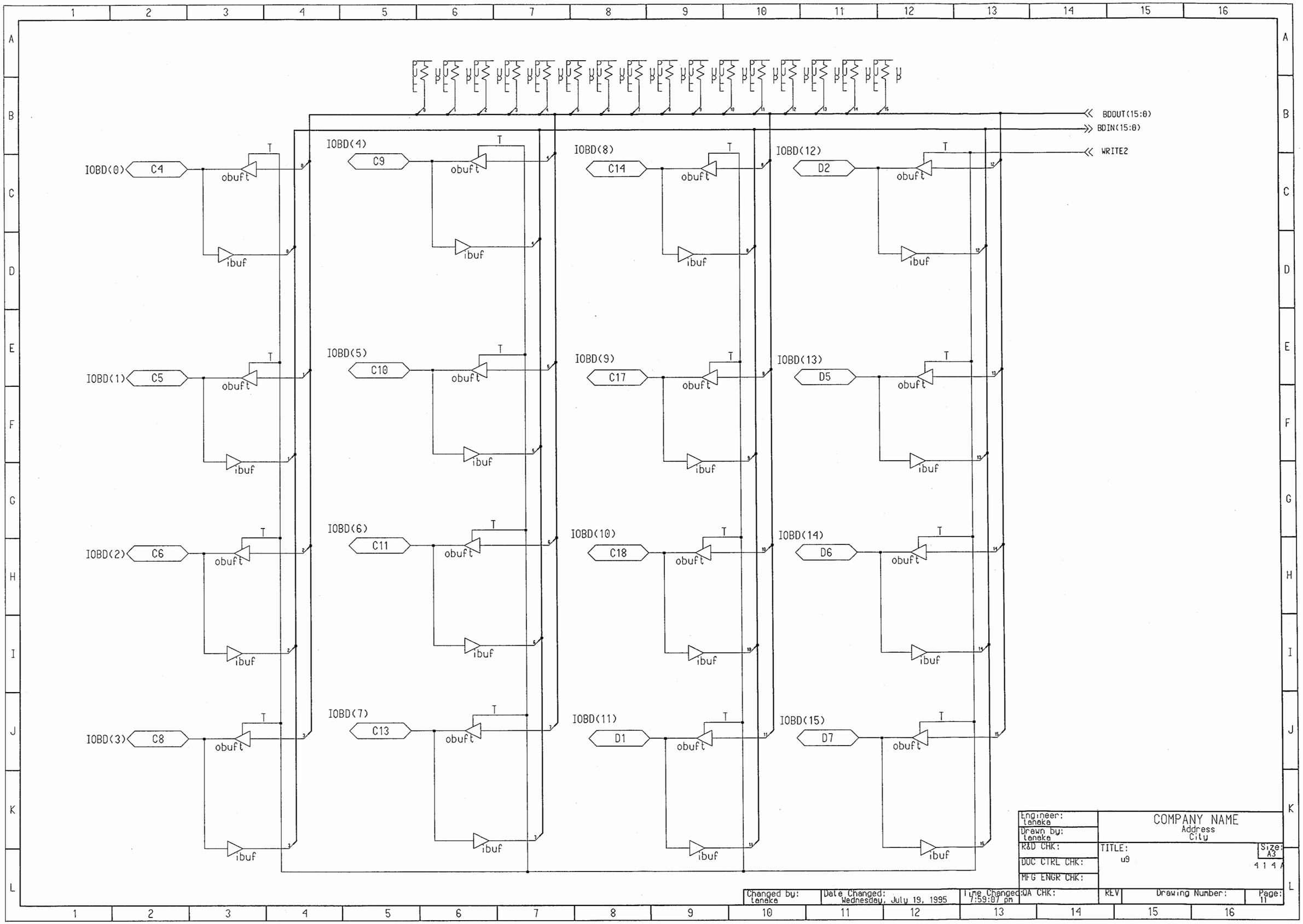

ICOMP_IN2(0)
F18
ICOMP_IN2(1)
G1
ICOMP_IN2(2)
G2
ICOMP_IN2(3)
G4
ICOMP_IN2(4)
G15
ICOMP_IN2(5)
G17
ICOMP_IN2(6)
G18
ICOMP_IN2(7)
H1
COMP_IN2(7:0)
ICOMP_IN3(0)
H4
ICOMP_IN3(1)
H15
ICOMP_IN3(2)
H16
ICOMP_IN3(3)
H17
ICOMP_IN3(4)
H18
ICOMP_IN3(5)
J1
ICOMP_IN3(6)
J2
ICOMP_IN3(7)
J3
COMP_IN3(7:0)
ifd
D
Q
R
C
MCLK
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
4 1 4 A
Changed by: tanaka
Date Changed: Wednesday, July 19, 1995
Time Changed: 6:17:19 pm
QA CHK:
REV
Drawing Number:
Page: 4

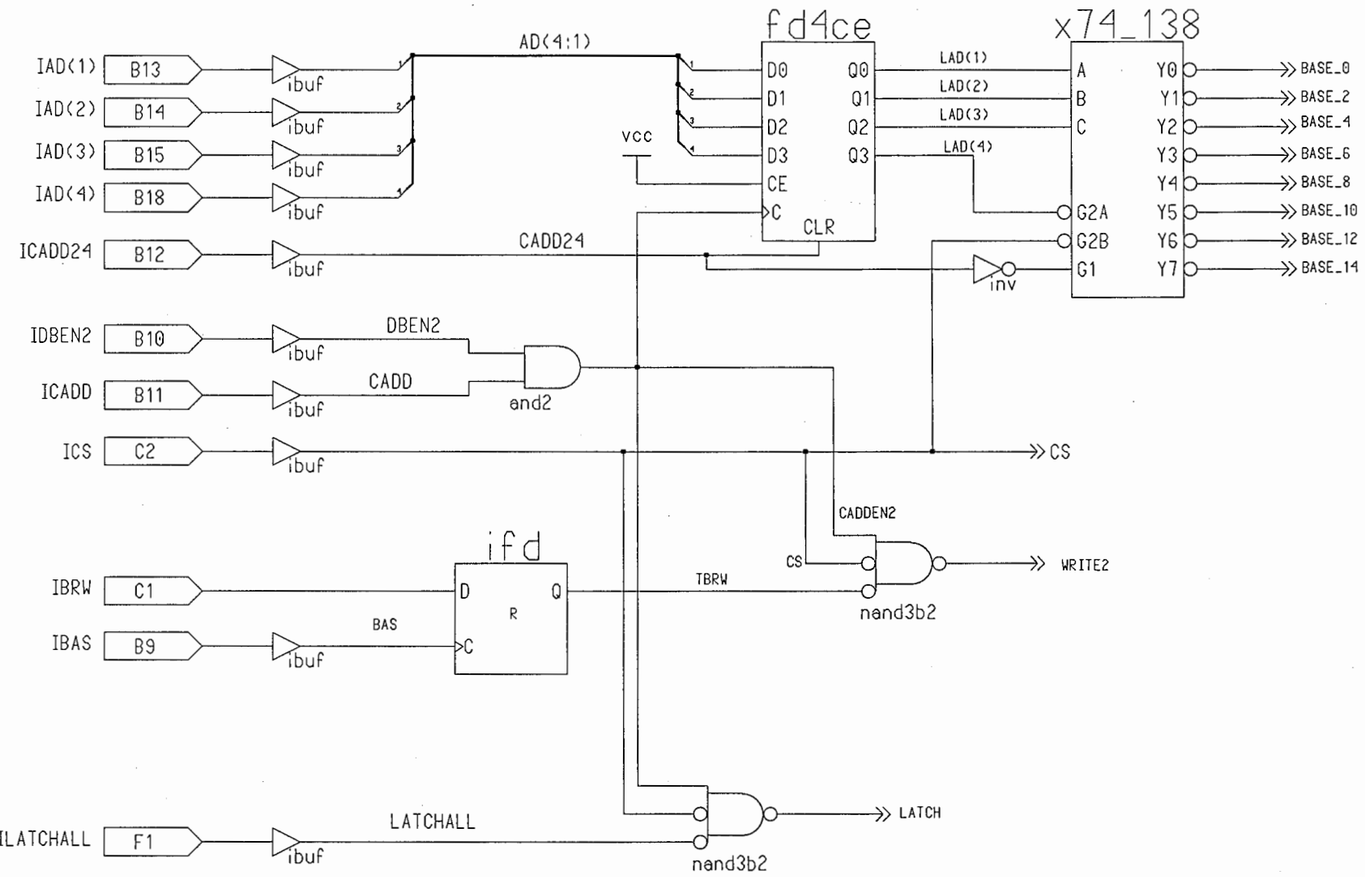

ICOMP_IN4(0)
K1
ICOMP_IN4(1)
K2
ICOMP_IN4(2)
K3
ICOMP_IN4(3)
K16
ICOMP_IN4(4)
K17
ICOMP_IN4(5)
K18
ICOMP_IN4(6)
L1
ICOMP_IN4(7)
L2
COMP_IN4(7:0)
ICOMP_IN5(0)
L15
ICOMP_IN5(1)
L16
ICOMP_IN5(2)
L17
ICOMP_IN5(3)
L18
ICOMP_IN5(4)
M1
ICOMP_IN5(5)
M2
ICOMP_IN5(6)
M4
ICOMP_IN5(7)
M15
COMP_IN5(7:0)
ifd
D
Q
R
C
MCLK
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
4 1 4 A
Changed by: tanaka
Date Changed: Wednesday, July 19, 1995
Time Changed: 6:20:53 pm
QA CHK:
REV
Drawing Number:
Page: 5


ICOMP_IN6(0)
N1
ICOMP_IN6(1)
N2
ICOMP_IN6(2)
N3
ICOMP_IN6(3)
N4
ICOMP_IN6(4)
N15
ICOMP_IN6(5)
N16
ICOMP_IN6(6)
N17
ICOMP_IN6(7)
N18
COMP_IN6(7:0)
ICOMP_IN7(0)
P3
ICOMP_IN7(1)
P4
ICOMP_IN7(2)
P15
ICOMP_IN7(3)
P16
ICOMP_IN7(4)
P17
ICOMP_IN7(5)
P18
ICOMP_IN7(6)
R1
ICOMP_IN7(7)
R2
COMP_IN7(7:0)
ifd
D
Q
R
C
MCLK
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
4 1 4 A
Changed by: taneka
Date Changed: Wednesday, July 19, 1995
Time Changed: 6:24:26 pm
QA CHK:
REV
Drawing Number:
Page: 6


COMP_SELECT(7:0)
OCOMP_SELECT(0)
V6
obuf
OCOMP_SELECT(1)
V7
obuf
OCOMP_SELECT(2)
V8
obuf
OCOMP_SELECT(3)
V9
obuf
OCOMP_SELECT(4)
V10
obuf
OCOMP_SELECT(5)
V15
obuf
OCOMP_SELECT(6)
V16
obuf
OCOMP_SELECT(7)
V2
obuf
DISABLE(7:0)
ODISABLE(0)
E3
obuf
ODISABLE(1)
F17
obuf
ODISABLE(2)
H3
obuf
ODISABLE(3)
J18
obuf
ODISABLE(4)
L4
obuf
ODISABLE(5)
M18
obuf
ODISABLE(6)
P2
obuf
ODISABLE(7)
R6
obuf
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE: u9
Size: A3
4 1 4 A
REV
Drawing Number:
Page: 8
Changed by: tanaka
Date Changed: Wednesday, July 19, 1995
Time Changed: 6:37:30 pm


IADD_IN(0)
IADD_IN(1)
IADD_IN(2)
IADD_IN(3)
IADD_IN(4)
IADD_IN(5)
IADD_IN(6)
IADD_IN(7)
ifd
R7
R8
R11
R12
R13
R14
R17
R18
D
Q
R
C
MCLK
ADD_IN(7:0)
U9, U10 Common
MAX_CH_SEL
B4
START
CMA_FIX
MAX_CH(3:0)
ofdt
T
T1
T2
T3
T5
OMAX_CH(0)
OMAX_CH(1)
OMAX_CH(2)
OMAX_CH(3)
ibuf
SEL_CH(3:0)
IMCLK
C3
bufgp
buf
obuf
U10
OMCLK
inv
IPHASE_CLK
B2
bufgs
PHASE_CLK
U9
OSTART
ISTART
U16
START
IMRST
B16
MRST
IBSYSCLK
B17
BSYSCLK
CMA_RST
V17
OCMA_RST
THRESH_OUT
U3
OTHRESH_OUT
CMA_OUT_EN
B7
IFIR_EN
V11
FIR_EN
IFFT1_EN
V12
FFT1_EN
IFFT2_EN
V13
FFT2_EN
ISPEC_EN
V14
SPEC_EN
ISNAP_SHOT
J16
fd
SNAP_SHOT
SNAP_1D
SNAP_2D
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
REV
Drawing Number:
Page: 10
Changed by: tanaka
Date Changed: Wednesday, July 19, 1995
Time Changed: 7:10:17 pm


BDOUT(15:0)
BDIN(15:0)
WRITE2
IOBD(0)
C4
IOBD(1)
C5
IOBD(2)
C6
IOBD(3)
C8
IOBD(4)
C9
IOBD(5)
C10
IOBD(6)
C11
IOBD(7)
C13
IOBD(8)
C14
IOBD(9)
C17
IOBD(10)
C18
IOBD(11)
D1
IOBD(12)
D2
IOBD(13)
D5
IOBD(14)
D6
IOBD(15)
D7
T
obuft
ibuf
Changed by: taneke
Date Changed: Wednesday, July 19, 1995
Time Changed: 7:59:07 pm
Engineer: taneke
Drawn by: taneke
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE:
u9
Size: A3
REV
Drawing Number:
4 1 4 A
Page: 11



Changed by: taneke	Date Changed: Wednesday, July 19, 1995	Time Changed: 8:02:27 pm	QA CHK:

Engineer: taneke	COMPANY NAME
Drawn by: taneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u9
MFG ENGR CHK:	Size: A3
	4 1 4 A
REV	Drawing Number:
	Page: 12

PHASE CLK GENERATOR

SELCTOR SAMPLE TIMING

DEFAULT
"10011100": 156
=100 cycle for 1 sample period

cb8cle
DECI_COUNT(7:0)
D(7:0)
Q(7:0)
SPEC_SAMPLE
vcc
L
CE
CEO
START
inv
C
TC
CLR
gnd

cb4cle
vcc
D0
D1
D2
D3
Q0
Q1
Q2
Q3
L
CE
CEO
gnd
MCLK
C
TC
CLR

fdc
D
Q
inv
PHASE_CLK
MCLK
inv
C
CLR

fd
START
inv
D
Q
MCLK
inv
C

fd
D
Q
C

START GENERATOR

"11001001": 201
=55 cycle for 1 sample period

cb8cle
D(7:0)
Q(7:0)
vcc
gnd
L
MCLK
CE
CEO
inv
C
START_RECLK
inv
START_OUT
TC
CLR
MRST

ns_u10
START
MCLK
SPEC_EN
FIR_EN
FFT2_EN
FFT1_EN
FIR_COEF_ADD(3:0)
OUT_EN(2:0)
OUT_TRIG(7:0)

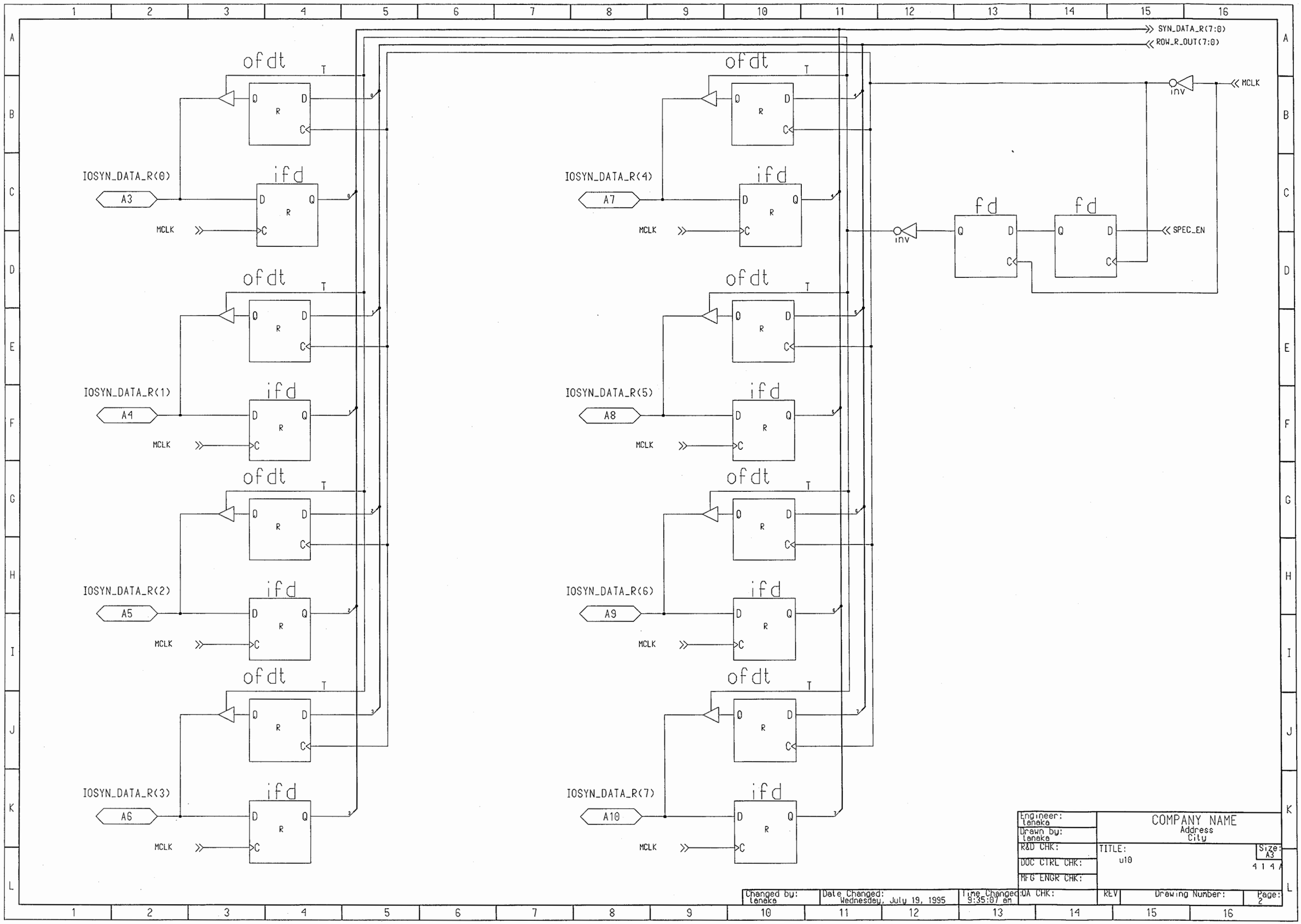
m2_1
THRESH
D0
FRST
D1
O
OUT_CONT
S0

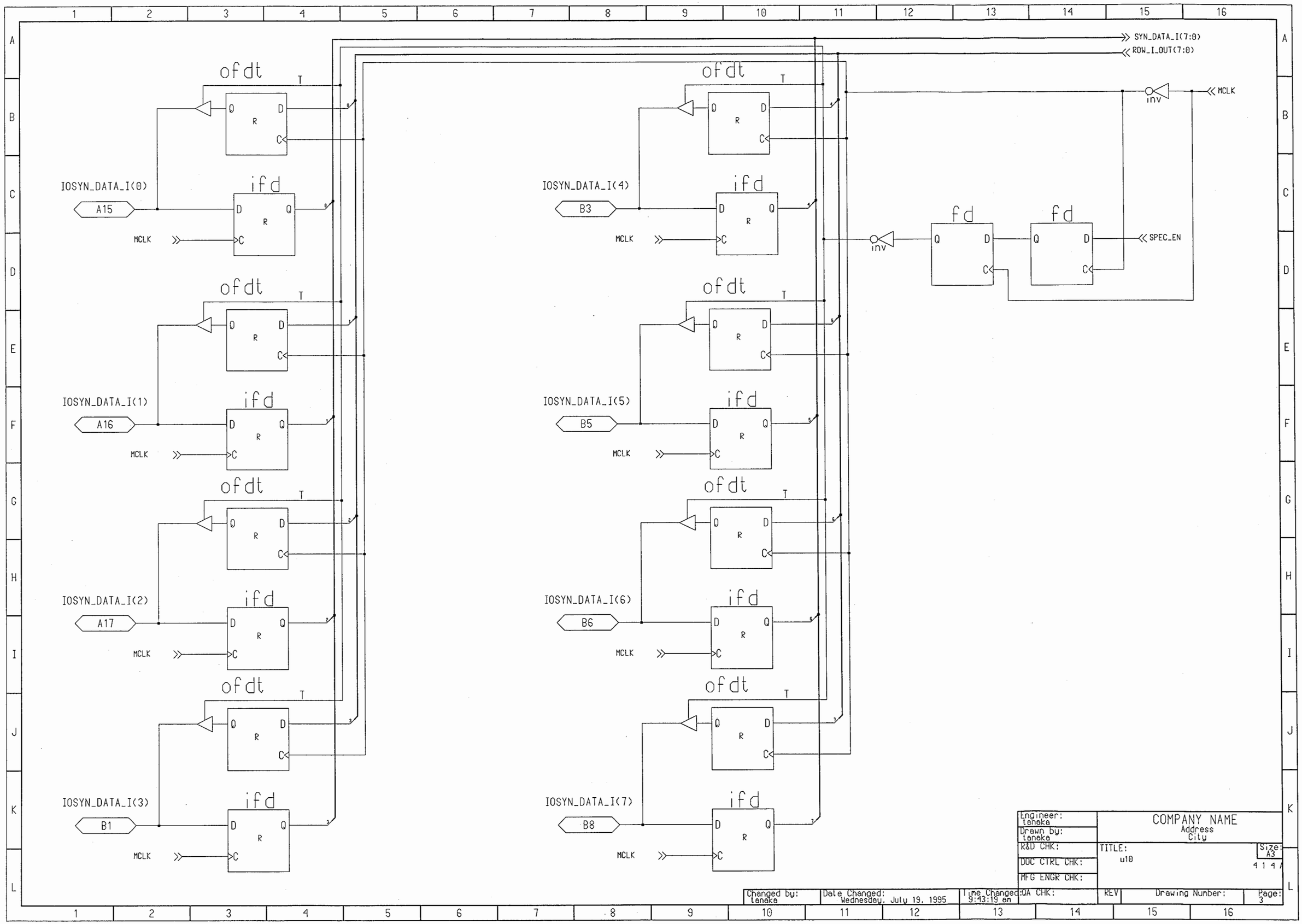
PHASE SHIFTER UNIT

SYN_DATA_R(7:0)
reg8
SRI_A(7:0)
DOUT(7:0)
DIN(7:0)
RADD(1:0)
ram48
WE
XACT5
SRO_A(7:0)
DIN_A(7:0)
DOUT(7:0)
DIN_B(7:0)
mux82_1
SEL
PMC_IN(7:0)
SYN_DATA_I(7:0)
reg8
SRI_B(7:0)
SYN_IN_ADD(1:0)
ram48
XACT5
SRO_B(7:0)
MCLK
inv
vcc
ZERO(15:0)
gnd
INTRIG
vcc
MCLK
MRST
reg8
MC_IN(7:0)
ss_u10
INWE
SYN_IO_SEL
ROW_RAM_Q_WE
ROW_RAM_I_WE
PH_ROM_SEL
OREG_TRIG
MC_RAM_WE
FRST
ADD_SUB
SYNC
CHM_SEL(1:0)
DATA_TRIG(2:0)
MC_RAM_ADD(1:0)
ROW_RAM_ADD(1:0)
SYN_IN_ADD(1:0)
START
MRST
MCLK
ND_SEL(1:0)
MC_RAM_PR_ADD(3:0)
phc_ram
I_IN(7:0)
O_IN(7:0)
ADD_IN(3:0)
XACT5
ADD_OUT(3:0)
SYN_ADDR(3:0)
ADD_ADDR(1:0)
XACT5
DIN_A(7:0)
DOUT(7:0)
DIN_B(7:0)
mux82_1
SEL
MCLK
MRST
reg8
MP_IN(7:0)
gnd
MRST
MCLK
ADD_SUB
SYNC
MC(7:0)
proc8_3
POUT(15:0)
MP(7:0)
LD_IN(15:0)
MRST
ACC_EN
MCLK
AS_CONT
LD MUL_REG(15:0)
ACC16_OUT(15:0)
15:7
OREG_TRIG
MCLK
MRST
reg8
OREG_OUT(7:0)
DOUT(7:0)
DIN(7:0)
RADD(1:0)
ram48
WE
XACT5
ROW_R_OUT(7:0)
MLREG_OUT(15:0)
DOUT(7:0)
DIN(7:0)
RADD(1:0)
ram48
WE
XACT5
ROW_I_OUT(7:0)
SYN_OUT_EN(3:0)
ROW_RAM_ADD(1:0)
ROW_RAM_I_WE
ROW_RAM_Q_WE

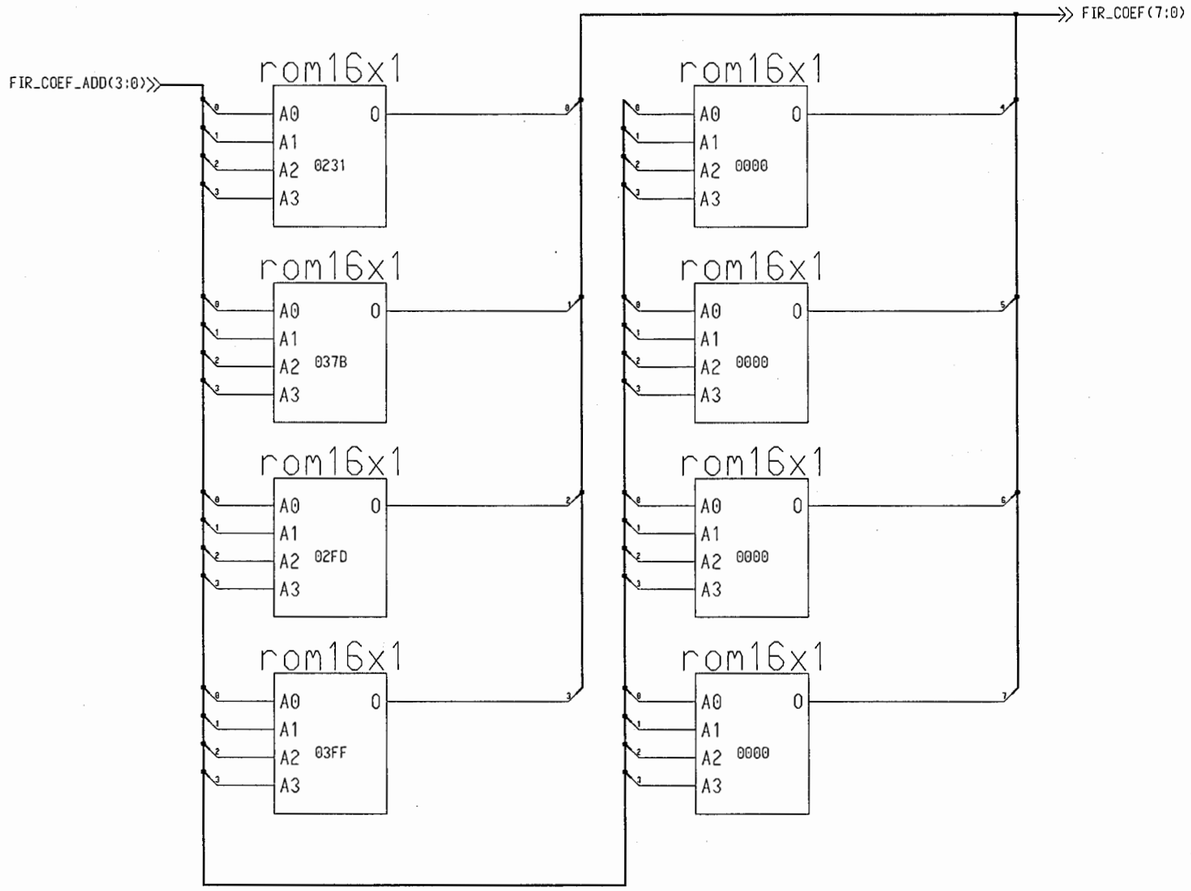
Engineer: tanaka	COMPANY NAME Address City		
Drawn by: tanaka			
R&D CHK:	TITLE: u10	Size: A3	
DOC CTRL CHK:			4 1 4 A
MFG ENGR CHK:			

Changed by:	Date Changed:	Time Changed:	QA CHK:	REV	Drawing Number:	Page:
tanaka	Tuesday, July 18, 1995	6:58:43 pm				1

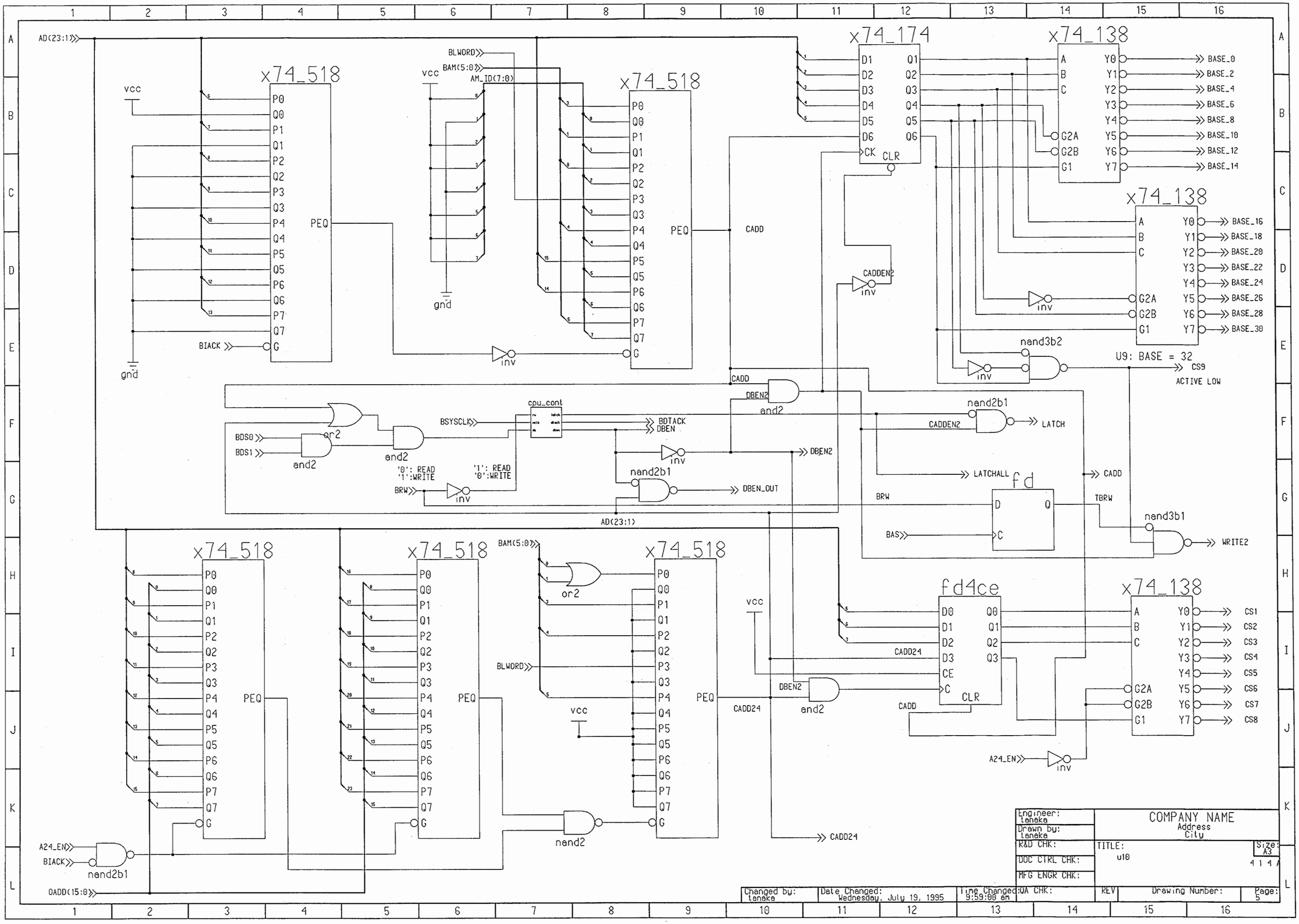

SYN_DATA_R(7:0)
ROW_R_OUT(7:0)
MCLK
SPEC_EN
inv
fd
ofdt
ifd
T
D
Q
C
R
IOSYN_DATA_R(0)
IOSYN_DATA_R(1)
IOSYN_DATA_R(2)
IOSYN_DATA_R(3)
IOSYN_DATA_R(4)
IOSYN_DATA_R(5)
IOSYN_DATA_R(6)
IOSYN_DATA_R(7)
A3
A4
A5
A6
A7
A8
A9
A10
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE: u10
Size: A3
DA CHK:
REV
Drawing Number:
Page: 2
Changed by: tanaka
Date Changed: Wednesday, July 19, 1995
Time Changed: 9:35:07 am



Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u10
MFG ENGR CHK:	Size: A3
Changed by: tanaka	Date Changed: Wednesday, July 19, 1995
Time Changed: 9:43:19 am	QA CHK:
REV	Drawing Number:
Page: 3	

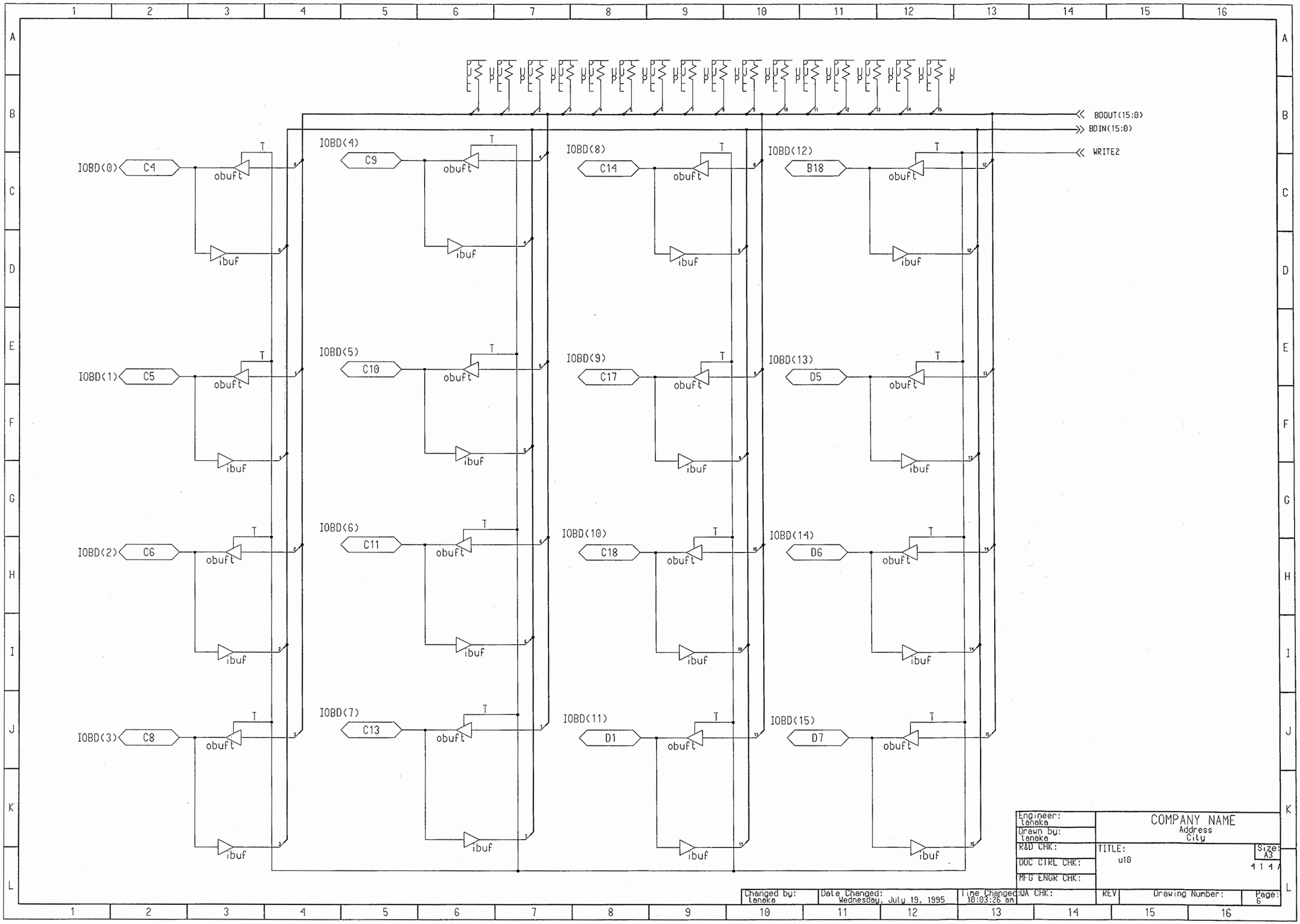


Engineer: tanaka	COMPANY NAME				
Drawn by: tanaka	Address				
R&D CHK:	City				
DOC CTRL CHK:	TITLE: u10	Size: A3			
MFG ENGR CHK:		4 1 4 A			
QA CHK:	REV	Drawing Number:	Page: 4		
Changed by: tanaka	Date Changed: Wednesday, July 19, 1995	Time Changed: 9:55:20 am			



Changed by: tenake	Date Changed: Wednesday, July 19, 1995	Time Changed: 9:59:00 am	QA CHK:	REV	Drawing Number:	Page: 5
---	---	---	---	---	---	---

Engineer: tenake	COMPANY NAME
Drawn by: tenake	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u10
MFG ENGR CHK:	Size: A3
	4 1 4 A

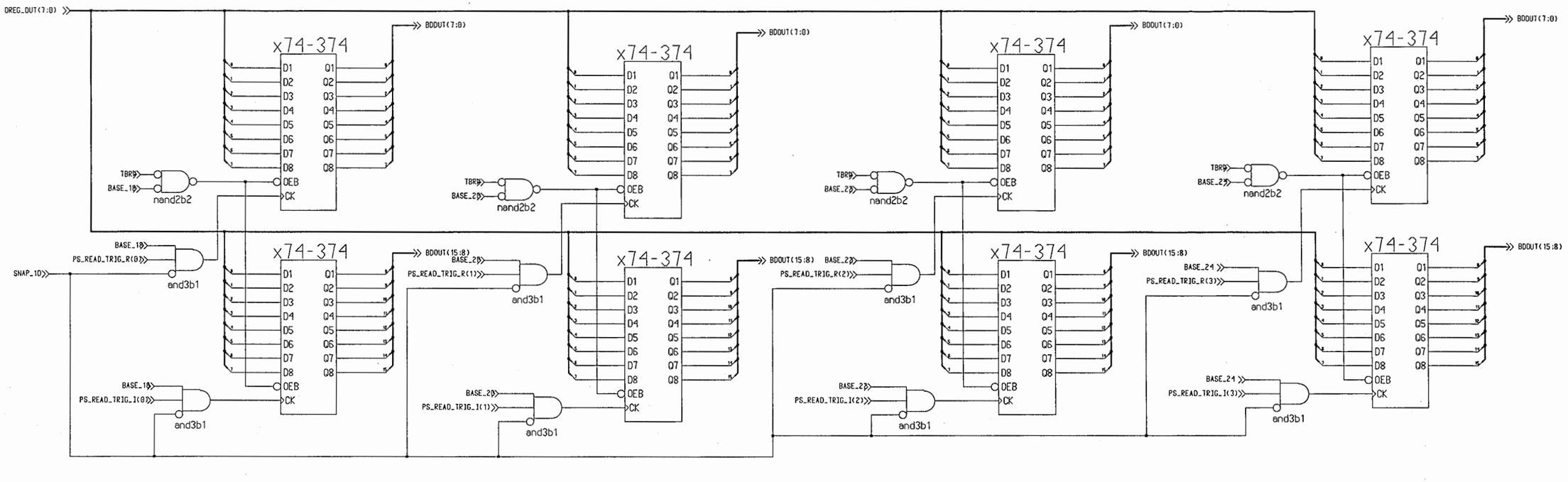

BDOUT(15:0)
BDIN(15:0)
WRITE2
IOBD(0)
C4
IOBD(1)
C5
IOBD(2)
C6
IOBD(3)
C8
IOBD(4)
C9
IOBD(5)
C10
IOBD(6)
C11
IOBD(7)
C13
IOBD(8)
C14
IOBD(9)
C17
IOBD(10)
C18
IOBD(11)
D1
IOBD(12)
B18
IOBD(13)
D5
IOBD(14)
D6
IOBD(15)
D7
T
obuft
ibuf
Changed by: taneke
Date Changed: Wednesday, July 19, 1995
Time Changed: 10:03:26 am
Engineer: taneke
Drawn by: taneke
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE: u10
Size: A3
REV
Drawing Number:
Page: 6

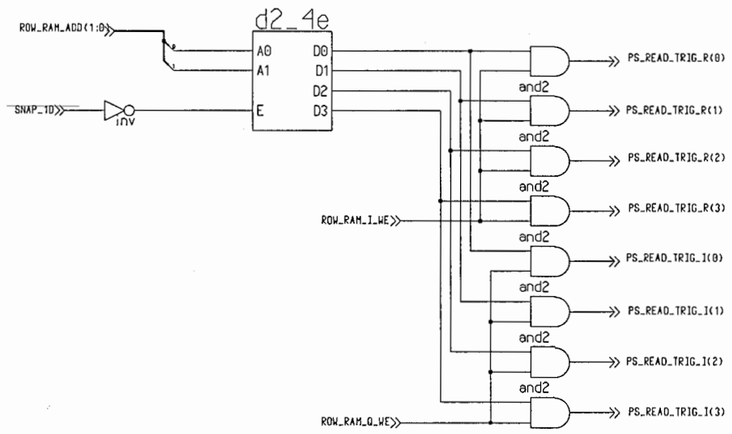
BASE +18 PS(0) REGISTER

BASE +20 PS(1) REGISTER

BASE +22 PS(2) REGISTER

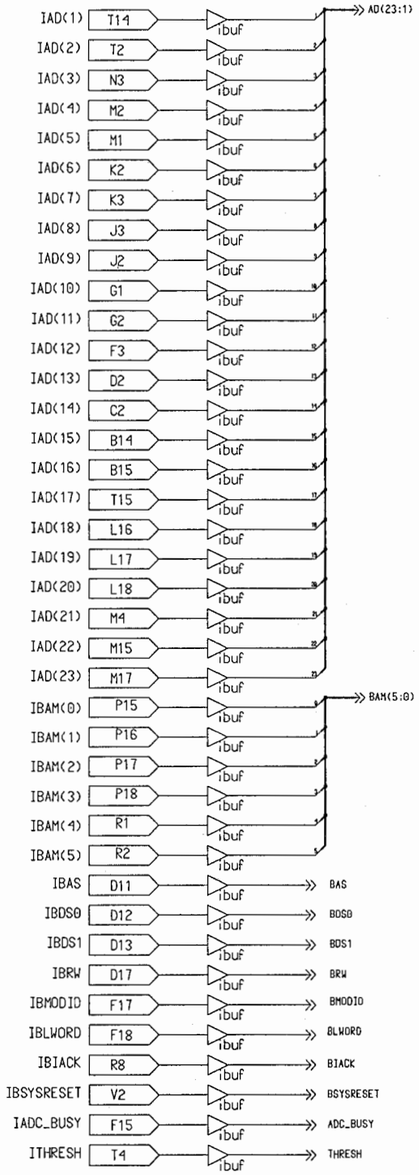
BASE +24 PS(3) REGISTER

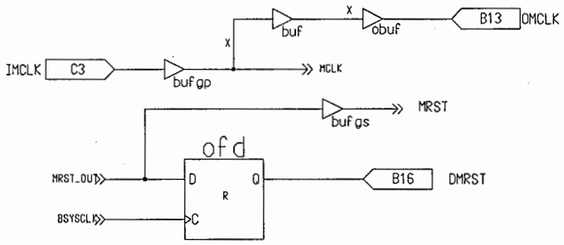


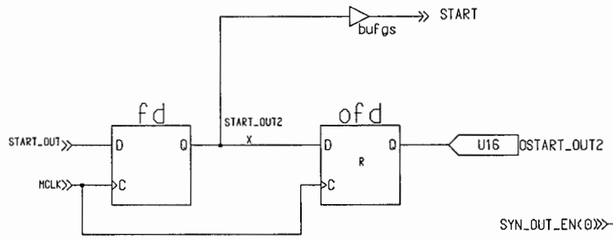


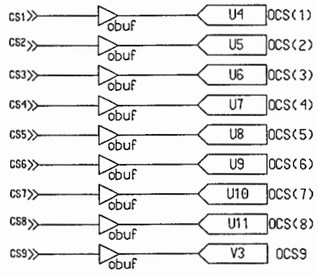
Changed by: tanaka	Date Changed: Sunday, January 14, 1996	Time Changed: 11:57:51 am	DA CHK:	REV	Drawing Number:	Page: 7

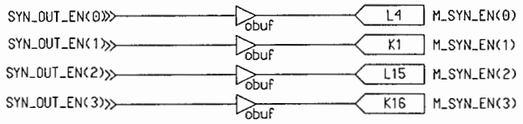
Engineer: tanaka	COMPANY NAME Address City	
Drawn by: tanaka		
R&D CHK:	TITLE: u10	Size: A2 4144
DOC CTRL CHK:		
MFG ENGR CHK:		

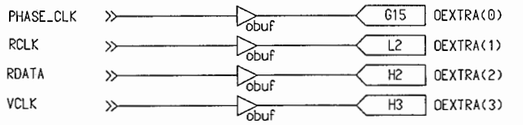


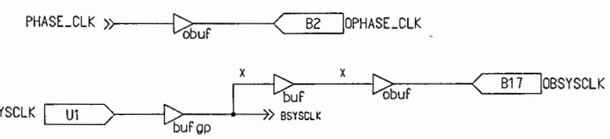


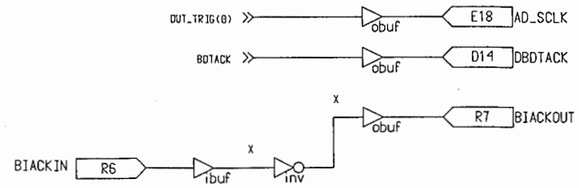


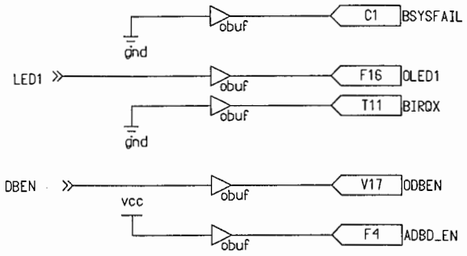


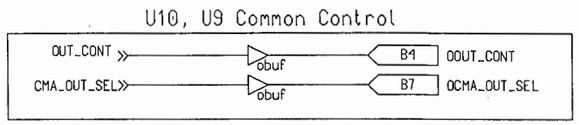


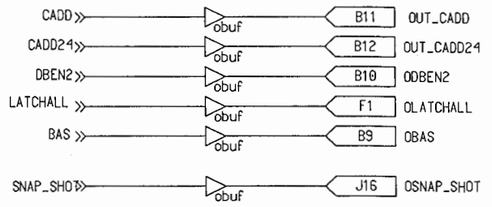






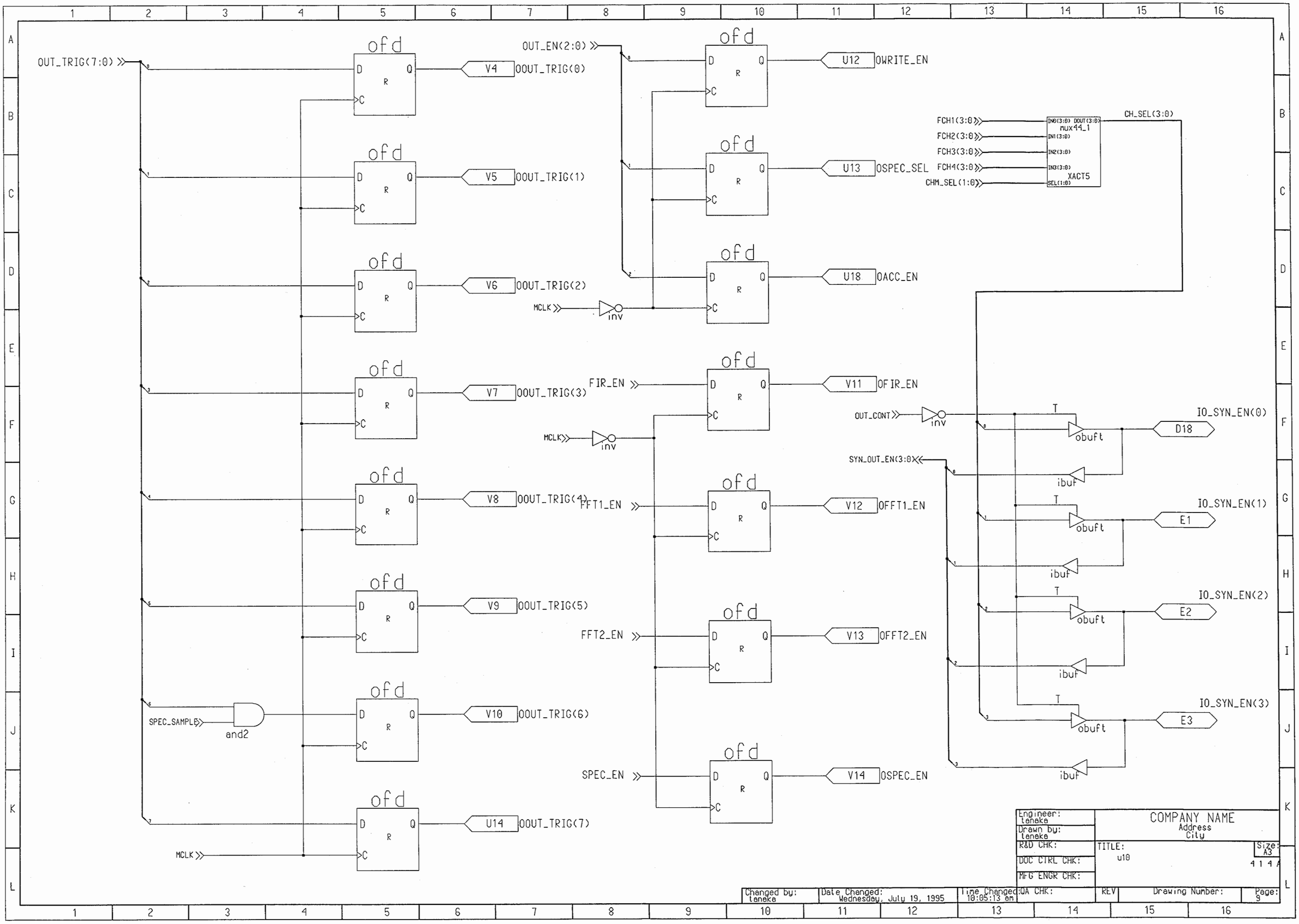


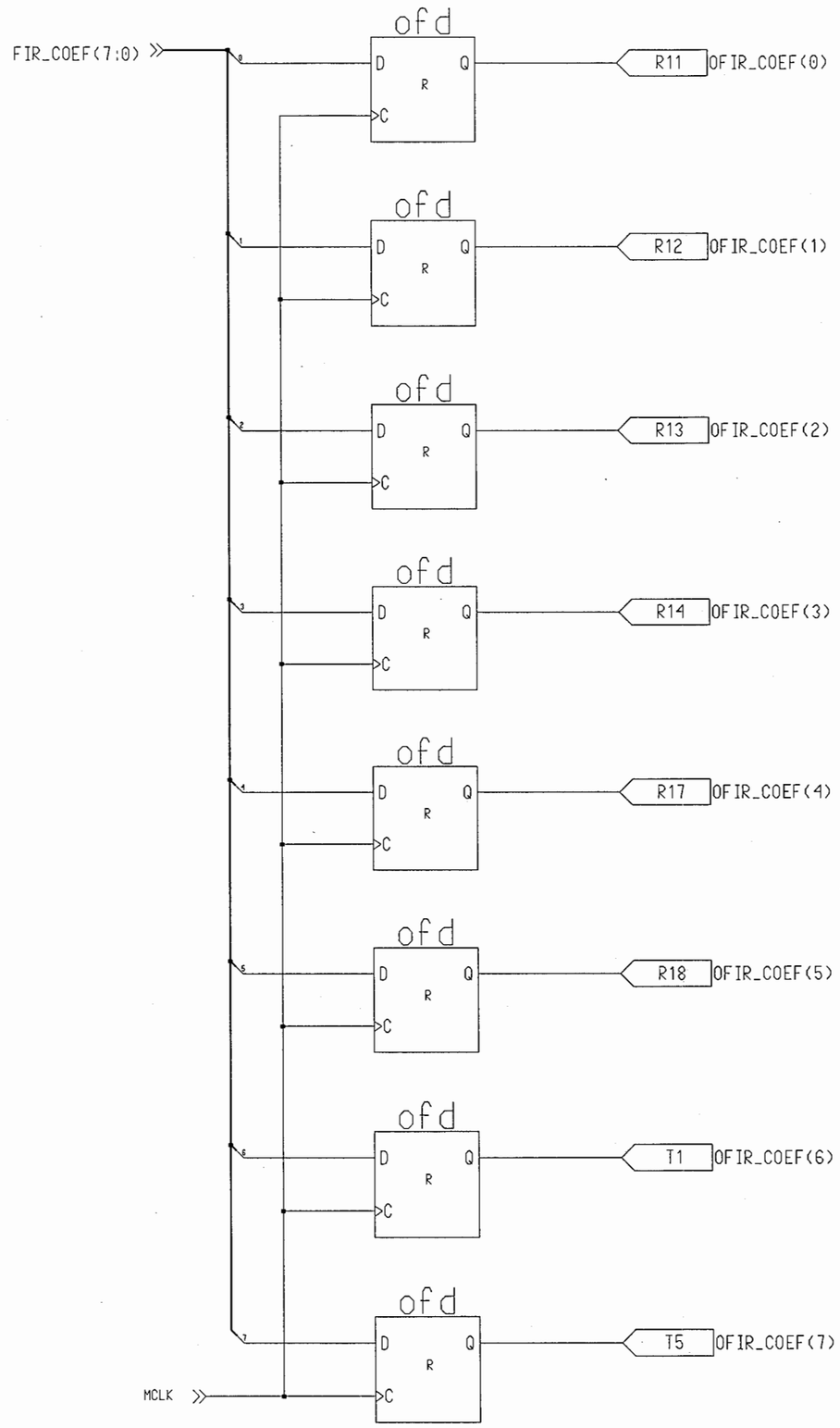




Engineer: Tanaka	COMPANY NAME		
Drawn by: Tanaka	Address		
R&D CHK:	City		
DOC CTRL CHK:	TITLE: u10	Size: A2	
MFG ENGR CHK:		1 1 1 /	
QA CHK:	REV	Drawing Number:	Page: 8

Changed by: Tanaka | Date Changed: Wednesday, July 19, 1995 | Time Changed: 11:20:49 am


OUT_TRIG(7:0)
ofd
D
Q
R
C
V4
OOUT_TRIG(0)
V5
OOUT_TRIG(1)
V6
OOUT_TRIG(2)
V7
OOUT_TRIG(3)
V8
OOUT_TRIG(4)
V9
OOUT_TRIG(5)
V10
OOUT_TRIG(6)
U14
OOUT_TRIG(7)
SPEC_SAMPLE
and2
MCLK
inv
OUT_EN(2:0)
U12
OWRITE_EN
U13
OSPEC_SEL
U18
OACC_EN
FIR_EN
V11
OFIR_EN
FFT1_EN
V12
OFFT1_EN
FFT2_EN
V13
OFFT2_EN
SPEC_EN
V14
OSPEC_EN
FCH1(3:0)
FCH2(3:0)
FCH3(3:0)
FCH4(3:0)
CHM_SEL(1:0)
IN0(3:0)
DOUT(3:0)
mux44_1
IN1(3:0)
IN2(3:0)
IN3(3:0)
SEL(1:0)
XACT5
CH_SEL(3:0)
OUT_CONT
T
obuft
ibuf
IO_SYN_EN(0)
D18
SYN_OUT_EN(3:0)
IO_SYN_EN(1)
E1
IO_SYN_EN(2)
E2
IO_SYN_EN(3)
E3
Engineer: taneka
Drawn by: taneka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
QA CHK:
COMPANY NAME
Address
City
TITLE:
u10
Size: A3
4 1 4 A
REV
Drawing Number:
Page: 9
Changed by: taneka
Date Changed: Wednesday, July 19, 1995
Time Changed: 10:05:13 am



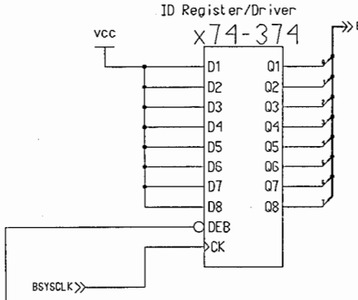
Engineer: laneke	COMPANY NAME
Drawn by: laneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u10
MFG ENGR CHK:	Size: A3
QA CHK:	REV / Drawing Number: / Page: 10

Changed by: laneke | Date Changed: Saturday, March 2, 1996 | Time Changed: 10:41:22 am

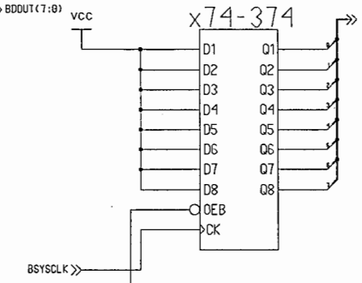
4 1 4 A

READ REGISTERS

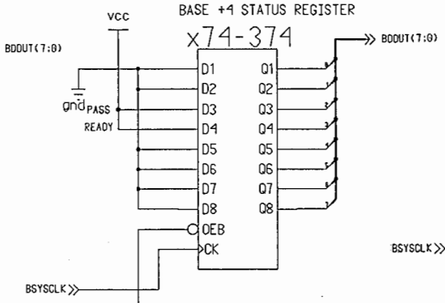
ID Register/Driver



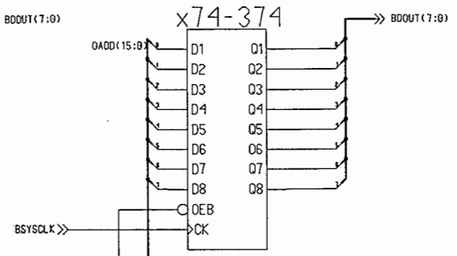
BASE +2 DEVICE TYPE REGISTER



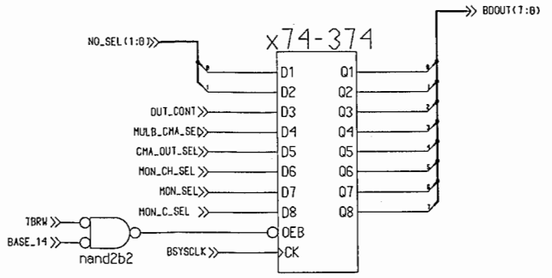
BASE +4 STATUS REGISTER

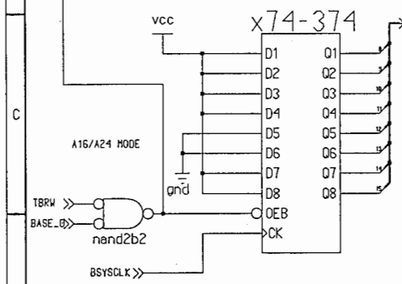


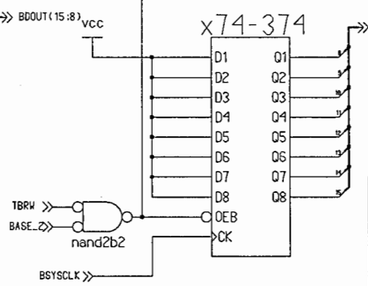
BASE +6 OFFSET REGISTER

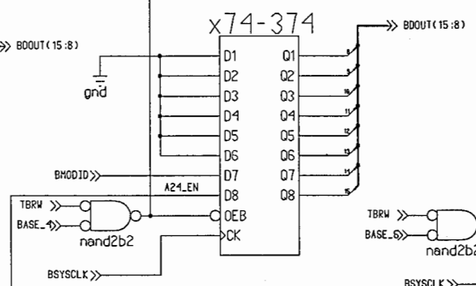


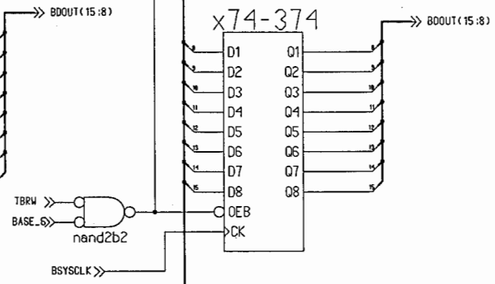
BASE +14 OUTPUT CONTROL READ BACK REGISTER





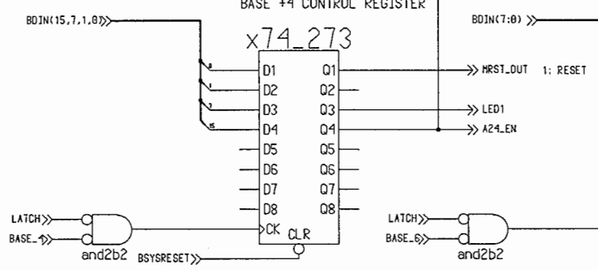




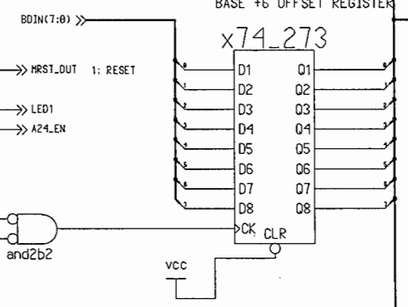


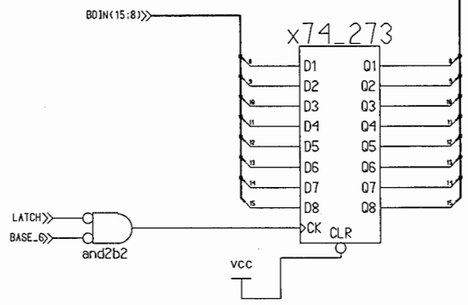
WRITE REGISTERS

BASE +4 CONTROL REGISTER

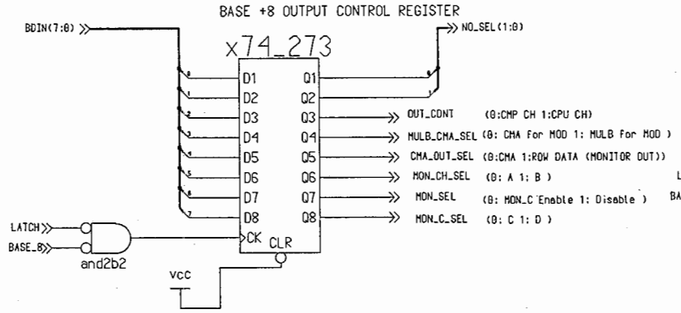


BASE +6 OFFSET REGISTER

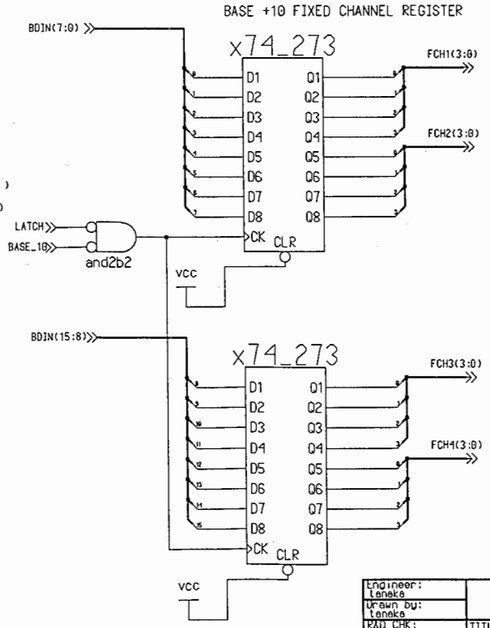




BASE +8 OUTPUT CONTROL REGISTER



BASE +10 FIXED CHANNEL REGISTER

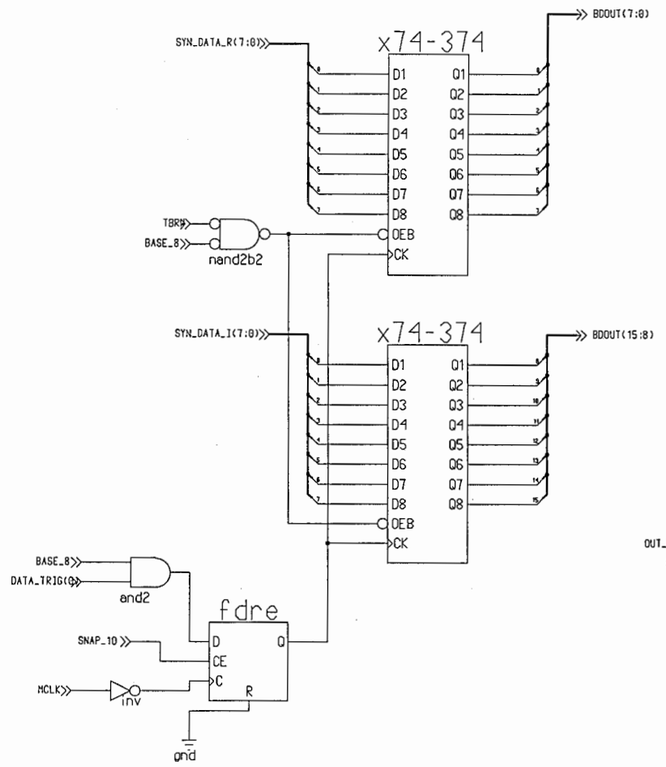


Engineer: Tanaka	COMPANY NAME		
Drawn by: Tanaka	Address		
R&D CHK:	City		
DOC CTRL CHK:	TITLE: u10	Size: A2	4141
MFG ENGR CHK:			

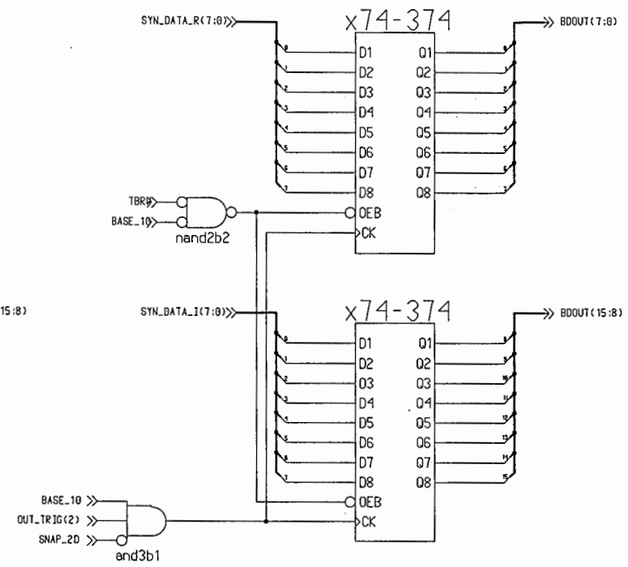
Changed by:	Date Changed:	Time Changed:	QA CHK:	REV	Drawing Number:	Page:
Tanaka	Wednesday, July 19, 1995	10:49:24 am				11

READ REGISTERS

BASE +8 SYN DATA REGISTER

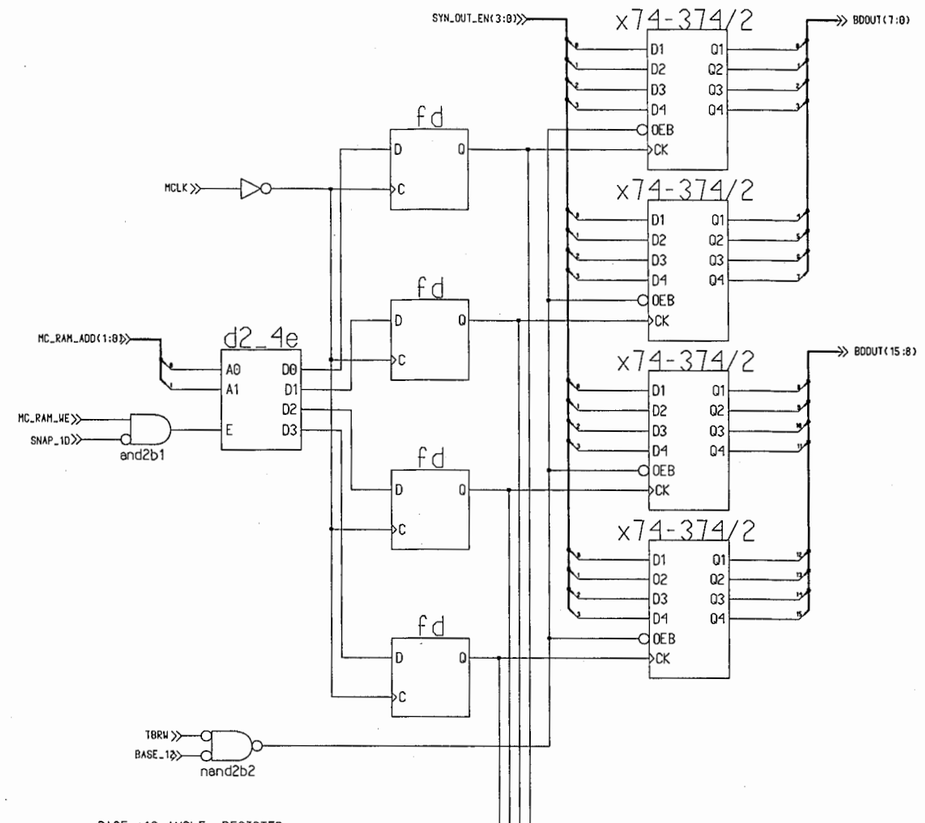


BASE +10 CMA DATA REGISTER

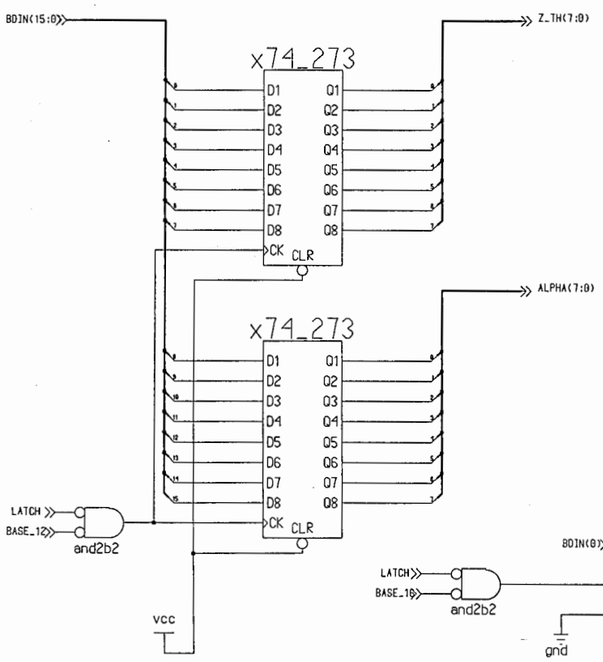


WRITE REGISTERS

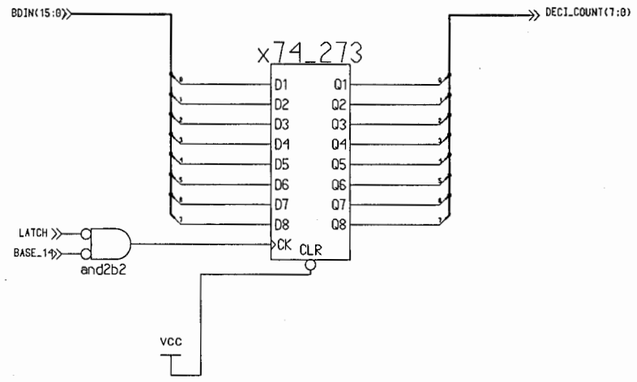
BASE +12 SEL CH DATA REGISTER



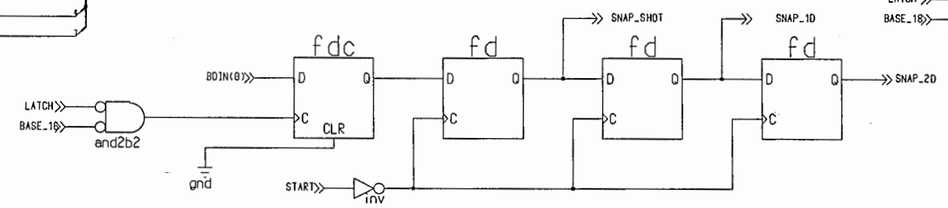
BASE +12 ALPHA/ZTH REGISTER



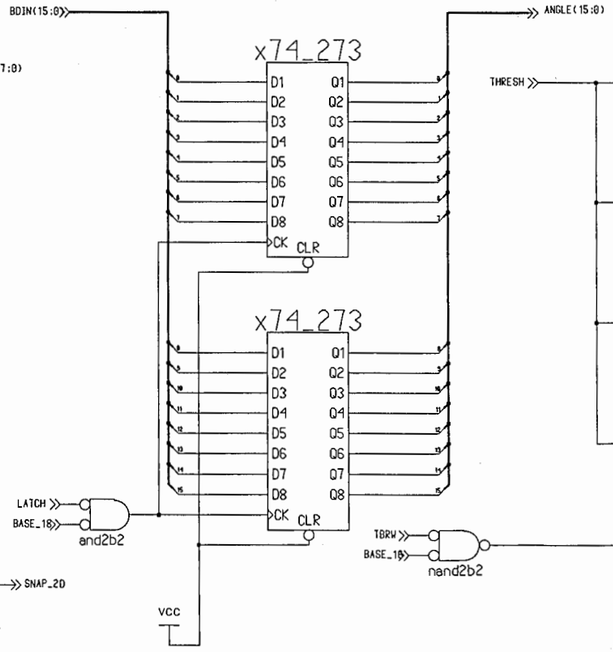
BASE +14 DECIMATION COUNT REGISTER



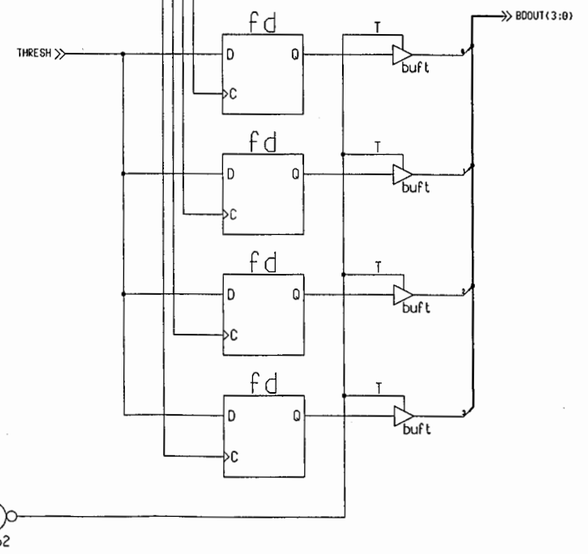
BASE +16 SNAP SHOT



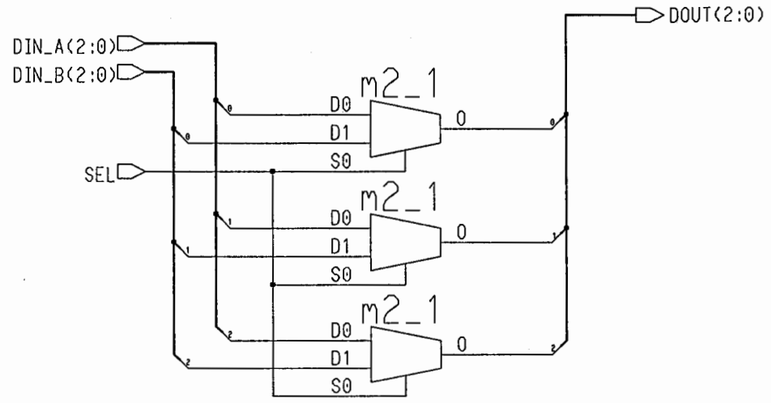
BASE +18 ANGLE REGISTER



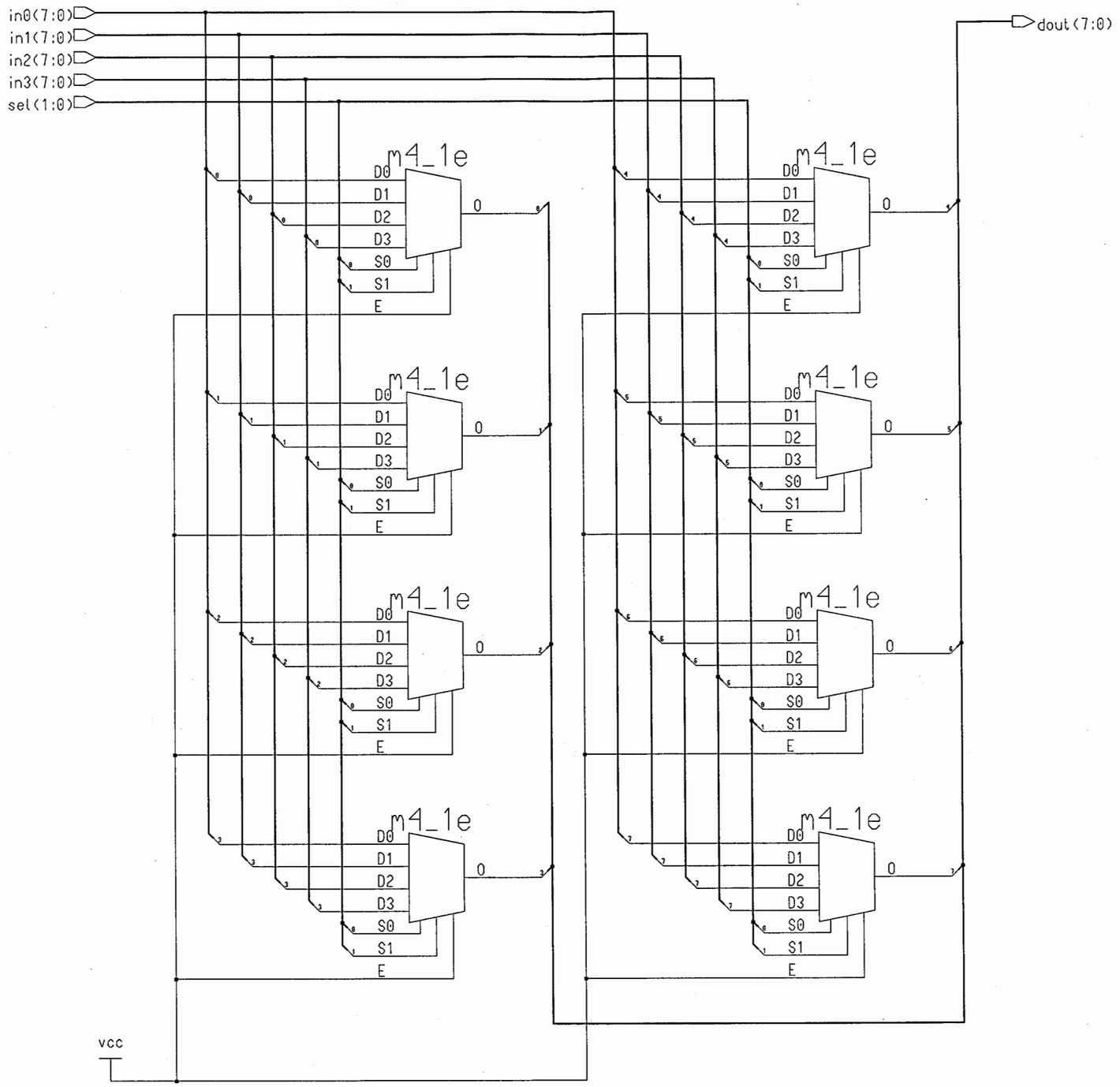
BASE +16 THRESH REGISTER



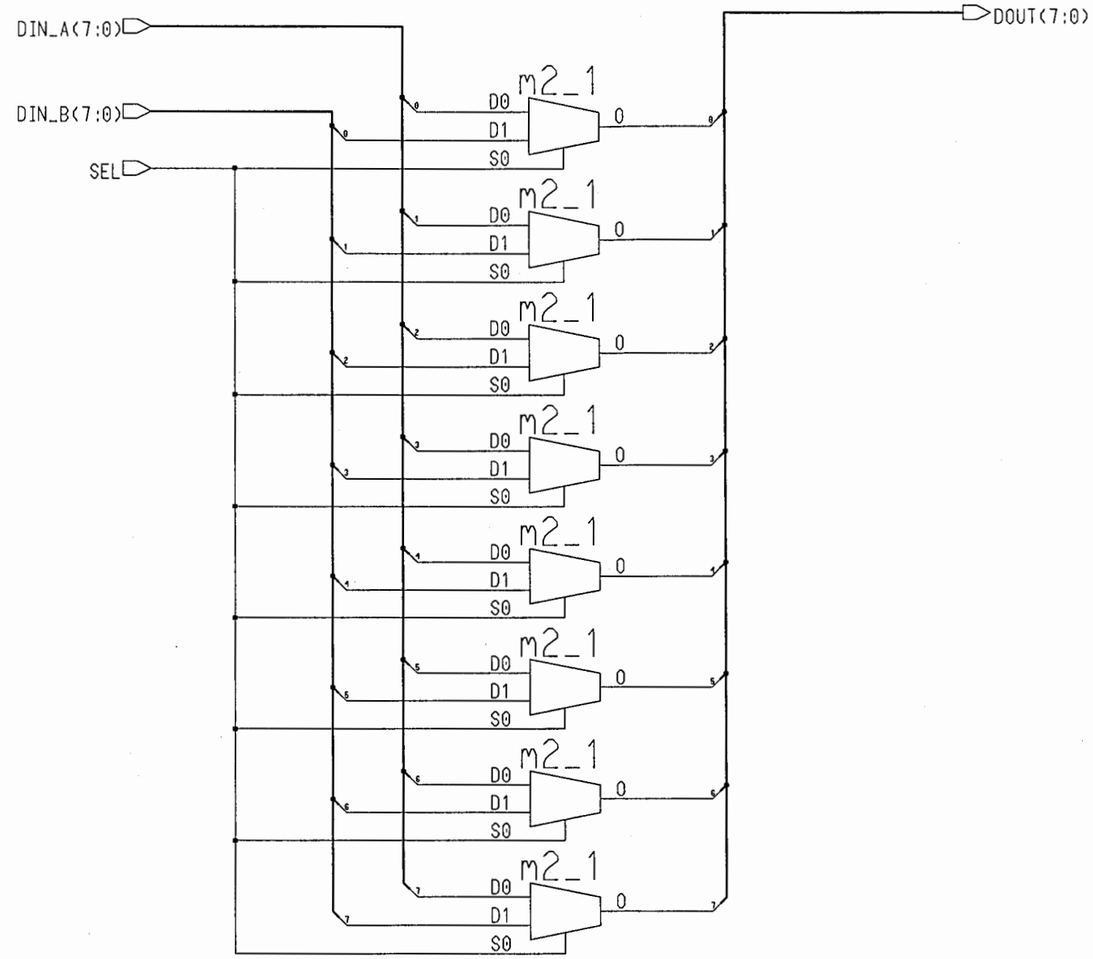
Engineer: Tanaka	COMPANY NAME
Drawn by: Tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: u18
MFG ENGR CHK:	Size: A2
Changed by: Tanaka	Date Changed: Wednesday, July 19, 1995
Time Changed: 10:49:59 am	QA CHK:
REV	Drawing Number:
Page: 12	



Engineer: tanaka	COMPANY NAME Address City				
Drawn by: tanaka					
R&D CHK:	TITLE: mux3_2_1	Size: A3			
DOC CTRL CHK:		4 1 4 A			
MFG ENGR CHK:					
Changed by: tanaka	Date Changed: Friday, July 14, 1995	Time Changed: 11:03:02 am	QA CHK:	REV	Drawing Number: / Page: 1

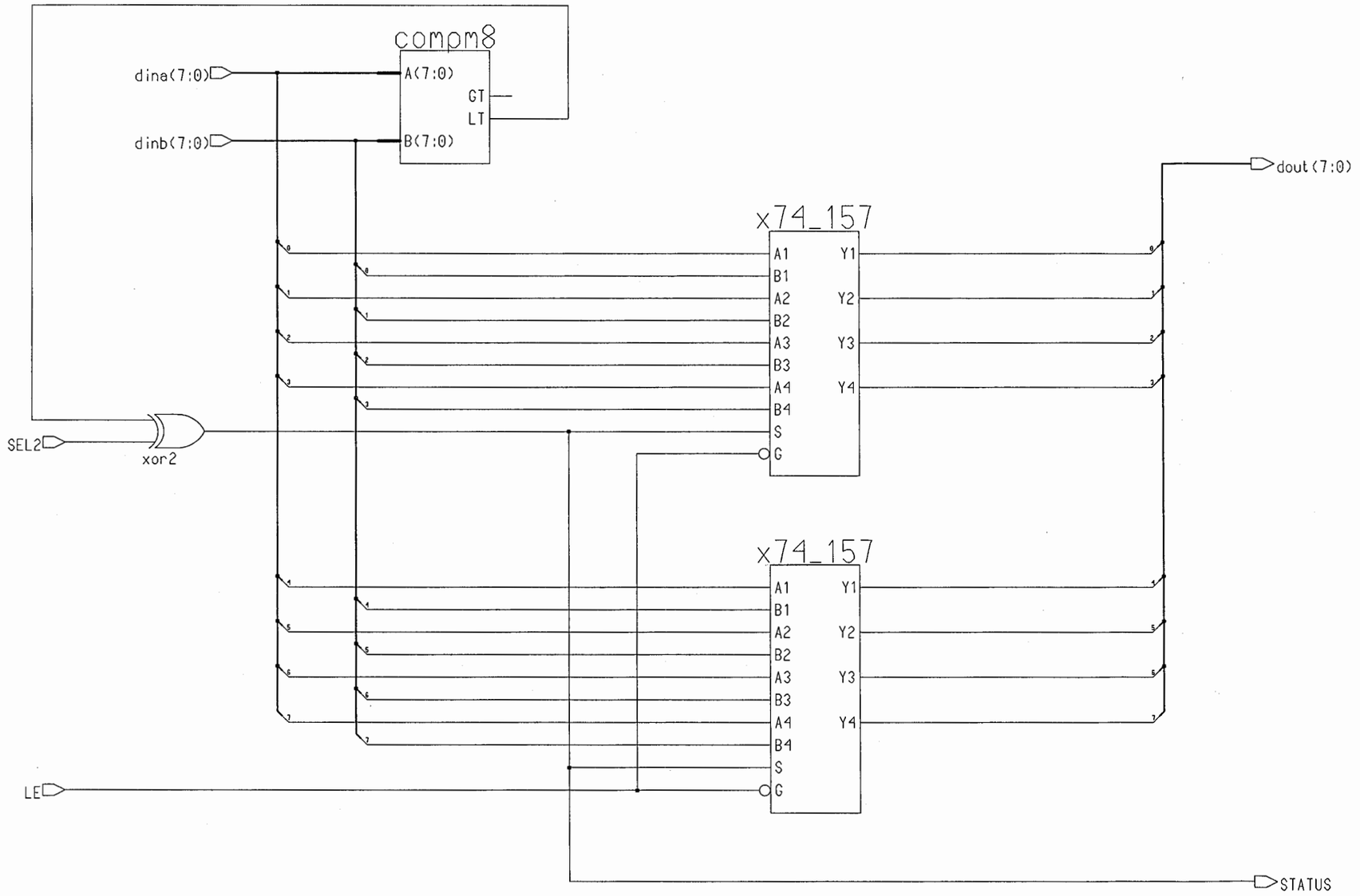


Engineer: tanaka	COMPANY NAME				
Drawn by: tanaka	Address				
R&D CHK:	City				
DOC CTRL CHK:	TITLE: mux84_1	Size: A3			
MFG ENGR CHK:					
Changed by: tanaka	Date Changed: Thursday, January 26, 1995	Time Changed: 2:16:28 pm	QA CHK:	REV	Drawing Number: 4 1 1 4 A	Page: 1



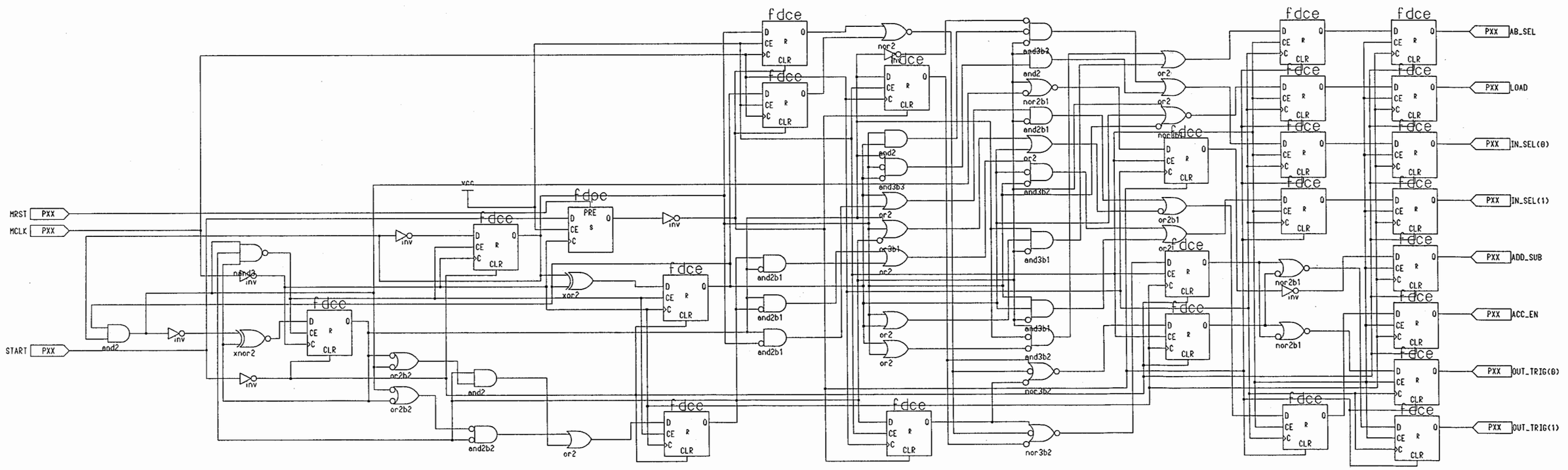
Engineer: tanaka	COMPANY NAME
Drawn by: tanaka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: mux82_1
MFG ENGR CHK:	Size: A3
QA CHK:	REV: 4 1 4 A

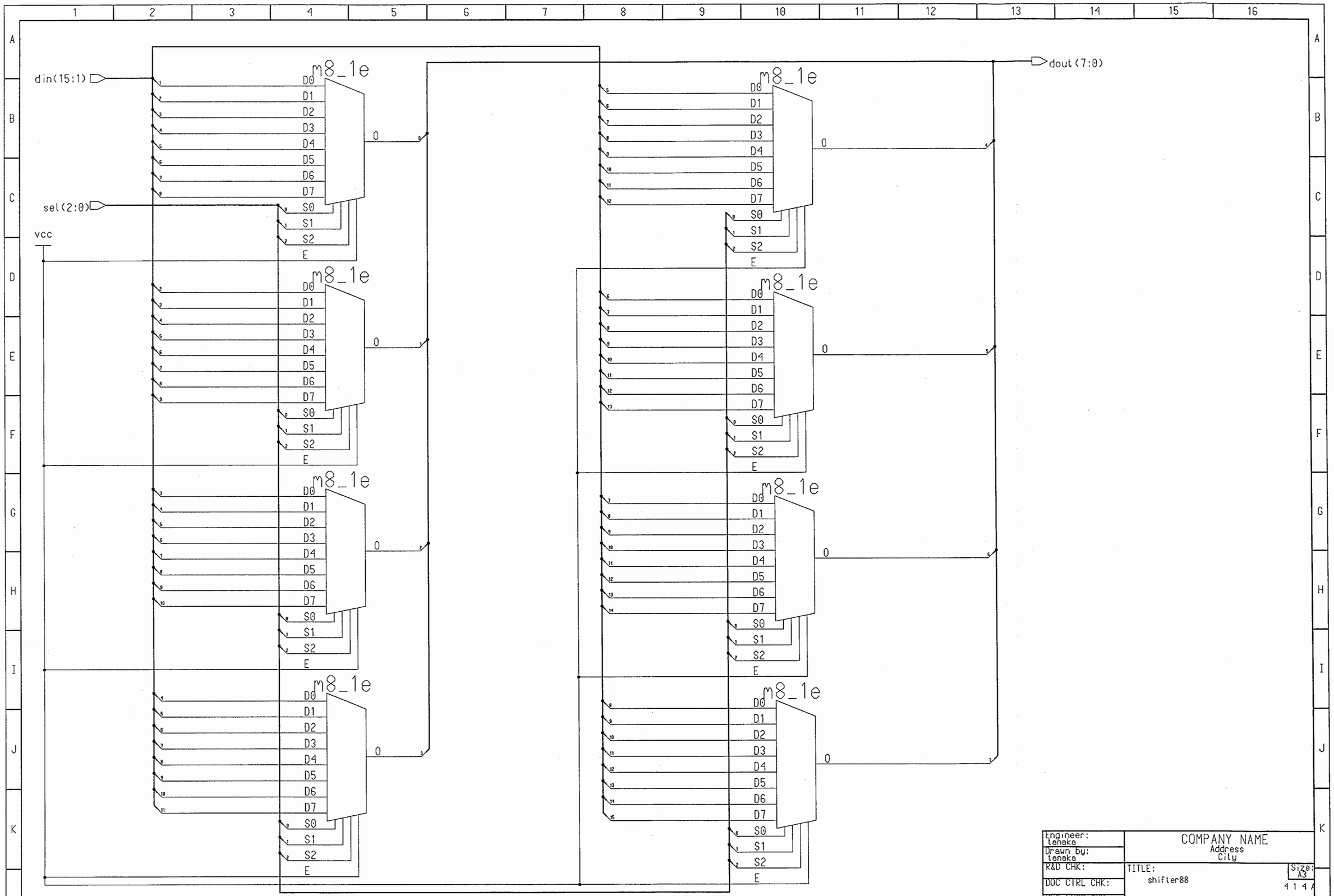
Changed by:	Date Changed:	Time Changed:	REV	Drawing Number:	Page:
tanaka	Wednesday, January 25, 1995	7:15:15 pm			1



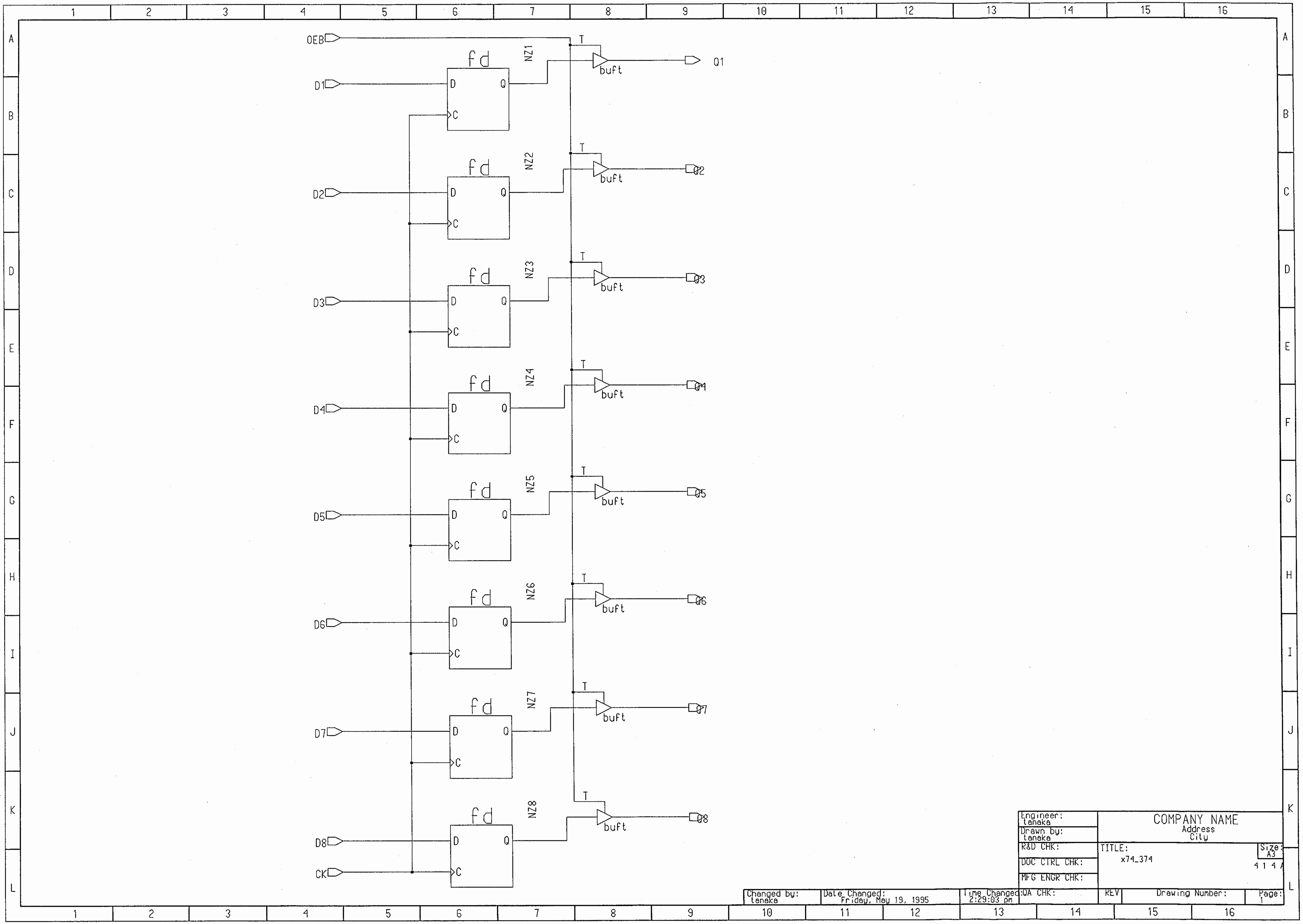
Engineer: taneka	COMPANY NAME
Drawn by: taneka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: comp_sel
MFG ENGR CHK:	Size: A3
QA CHK:	REV
Changed by: taneka	Drawing Number:
Date Changed: Wednesday, June 7, 1995	Page: 1
Time Changed: 7:01:43 pm	

sel_cont

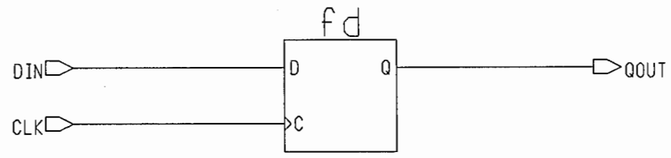

MRST
MCLK
START
PXX
vcc
fdce
fdpe
PRE
inv
and2
and3
xnor2
or2b2
and2b2
and2b1
and3b3
or3b1
and3b2
nor2
nor2b1
nor3b2
or2b1
AB_SEL
LOAD
IN_SEL(0)
IN_SEL(1)
ADD_SUB
ACC_EN
OUT_TRIG(0)
OUT_TRIG(1)



Changed by: tanaka	Date Changed: Wednesday, September 20, 1995	Time Changed: 10:49:40 am	QA CHK:	REV	Drawing Number:	Page: 1
Engineer: tanaka	COMPANY NAME Address City					
Drawn by: tanaka						
R&D CHK:	TITLE: shifter88	Size: A3				
DOC CTRL CHK:		4 1 4 A				
MFG ENGR CHK:						

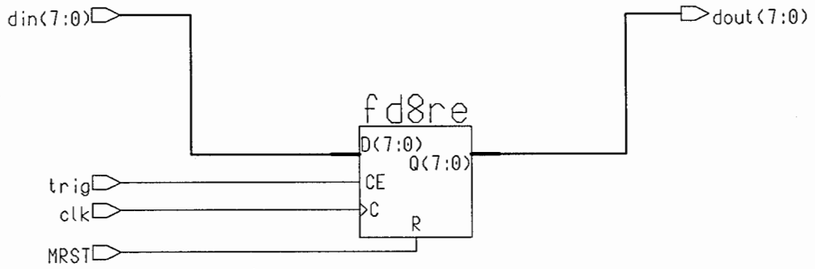


OEB
D1
D2
D3
D4
D5
D6
D7
D8
CK
fd
D
Q
C
NZ1
NZ2
NZ3
NZ4
NZ5
NZ6
NZ7
NZ8
T
buft
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE: x74_374
Size: A3
Changed by: tanaka
Date Changed: Friday, May 19, 1995
Time Changed: 2:29:03 pm
QA CHK:
REV
Drawing Number:
Page: 1

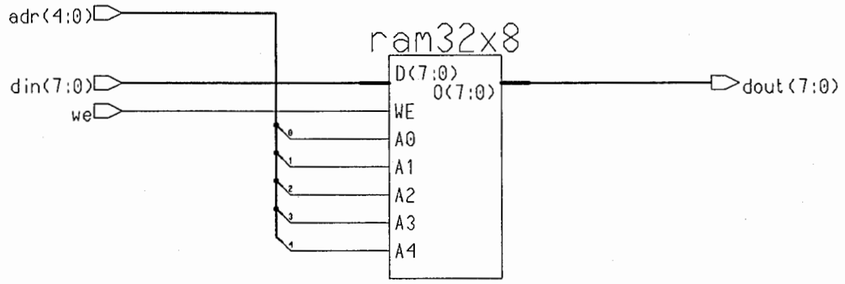


Changed by: tanaka	Date Changed: Wednesday, June 7, 1995	Time Changed: 6:38:14 pm			

Engineer: tanaka	COMPANY NAME				
Drawn by: tanaka	Address				
R&D CHK:	City				
DOC CTRL CHK:	TITLE: ff	Size: A3			
MFG ENGR CHK:		4 1 4 A			
QA CHK:	REV	Drawing Number:	Page: 1		



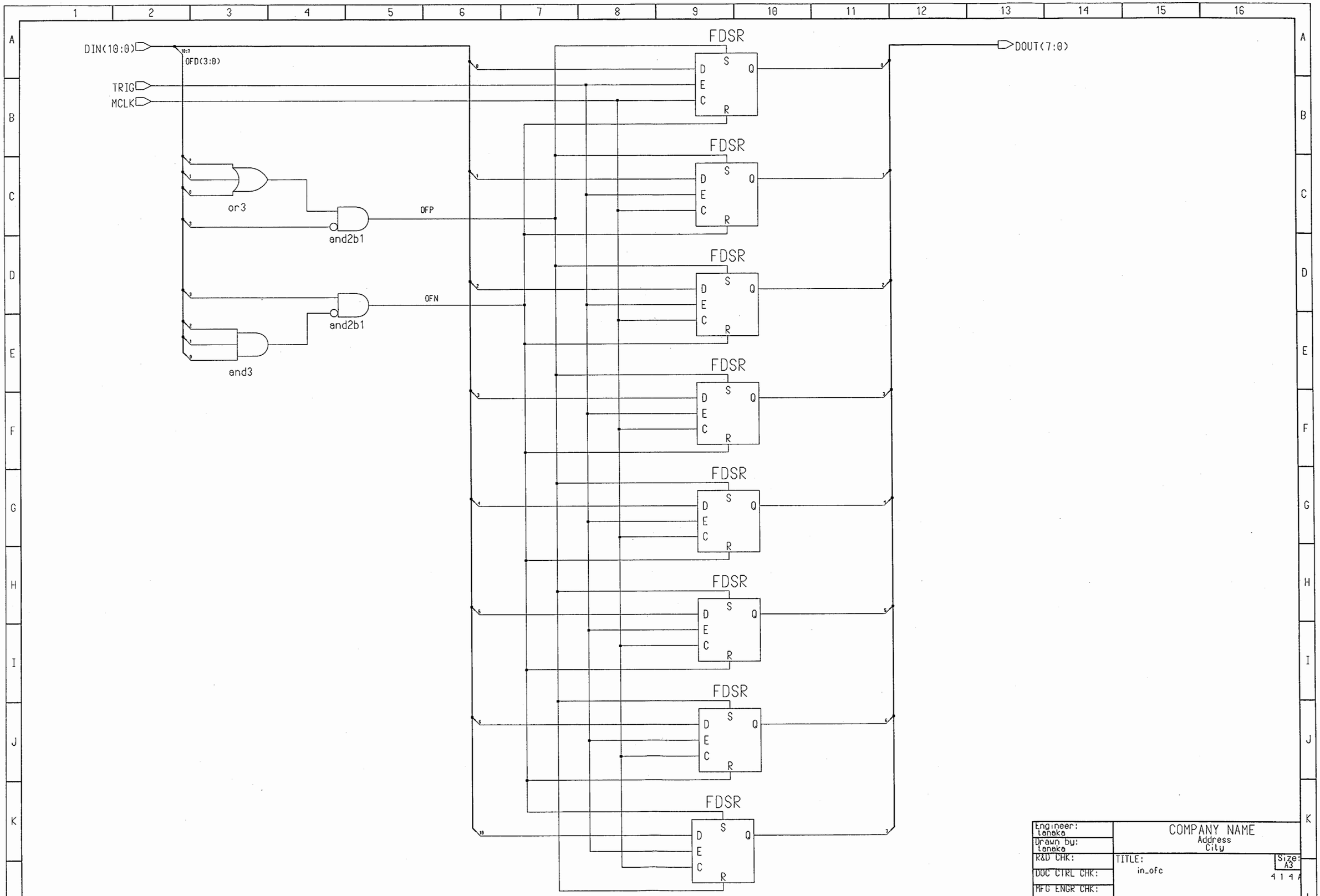
Engineer: tanaka	COMPANY NAME Address City				
Drawn by: tanaka					
R&D CHK:	TITLE: reg8	Size: A3			
DOC CTRL CHK:		4 1 4 A			
MFG ENGR CHK:					
Changed by: tanaka	Date Changed: Wednesday, June 7, 1995	Time Changed: 6:18:07 pm	QA CHK:	REV	Drawing Number: / Page: 1

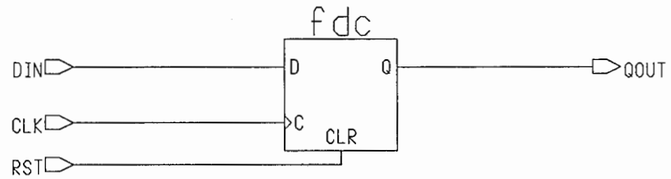


Changed by: laneke	Date Changed: Thursday, July 6, 1995	Time Changed: 6:15:52 pm	QA CHK:	REV	Drawing Number:

Engineer: laneke	COMPANY NAME
Drawn by: laneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: ram328
MFG ENGR CHK:	Size: A3
	4 1 4 A

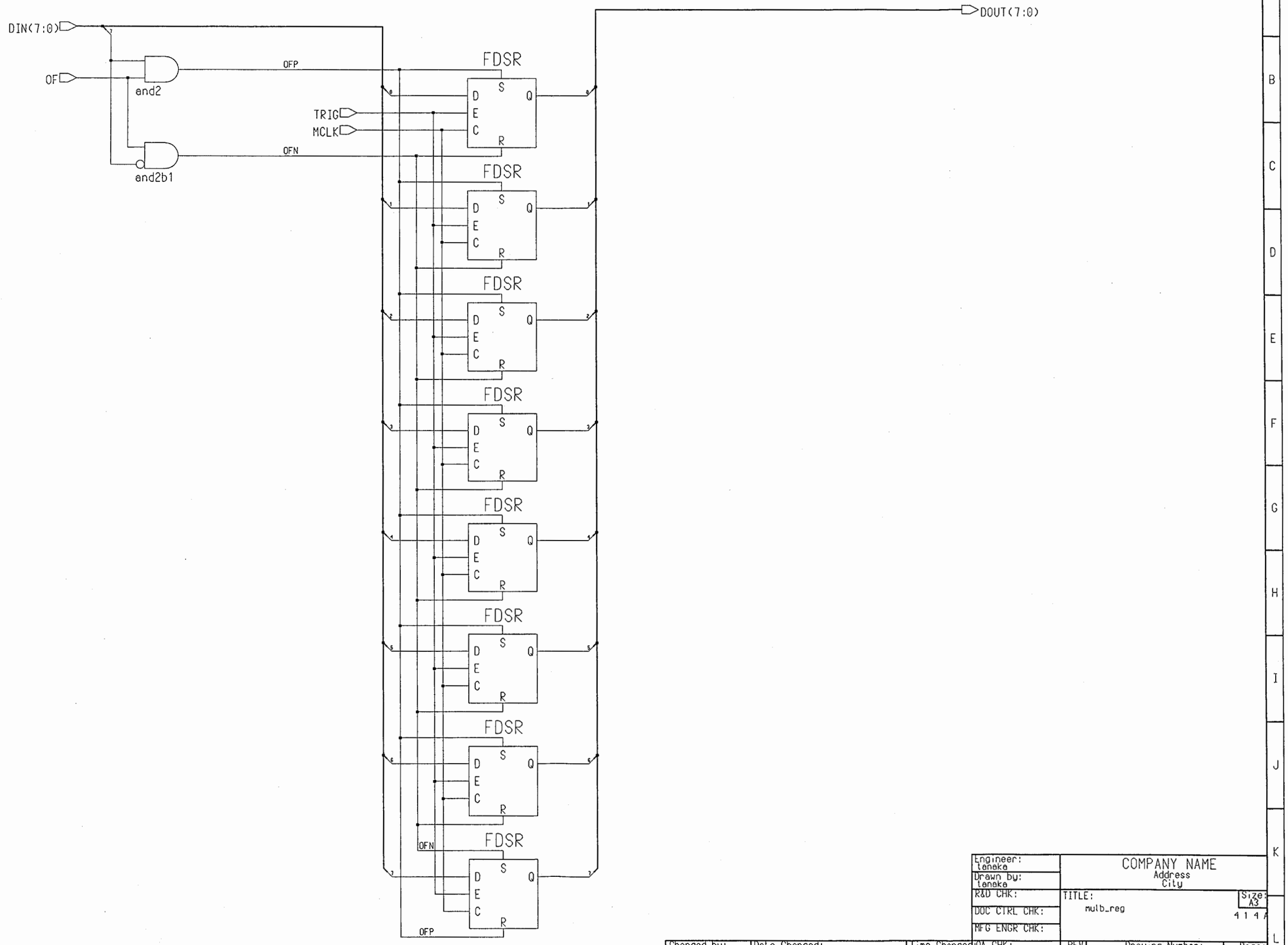
Page: 1


DIN(10:0)
OFD(3:0)
TRIG
MCLK
or3
and2b1
OFP
and2b1
OFN
and3
FDSR
FDSR
FDSR
FDSR
FDSR
FDSR
DOUT(7:0)
Engineer: tanaka
Drawn by: tanaka
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
in_ofc
Size: A3
Changed by: tanaka
Date Changed: Thursday, May 11, 1995
Time Changed: 7:28:08 pm
QA CHK:
REV
Drawing Number:
Page: 1



Changed by: tanaka	Date Changed: Wednesday, December 13, 1995	Time Changed: 8:24:05 pm				

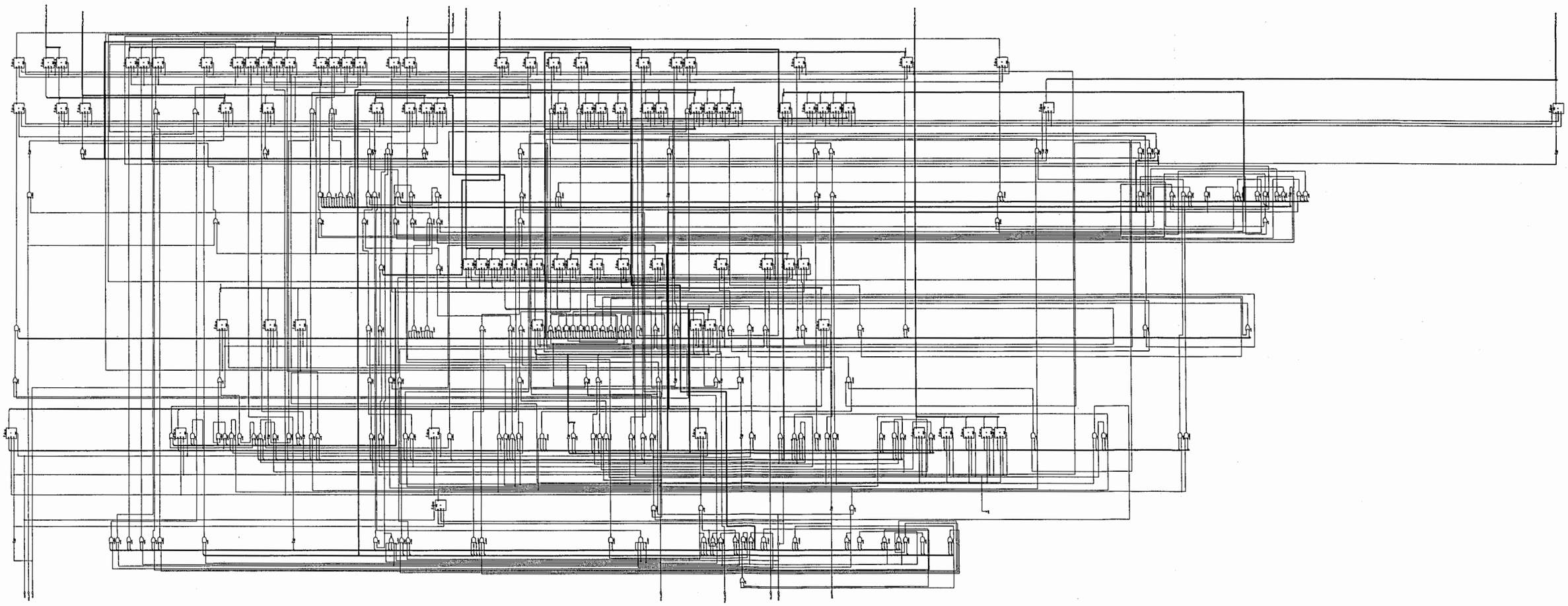
Engineer: tanaka	COMPANY NAME Address City		
Drawn by: tanaka			
R&D CHK:	TITLE: ffdc	Size: A3	
DOC CTRL CHK:		4 1 4 A	
MFG ENGR CHK:			
QA CHK:	REV	Drawing Number:	Page: 1

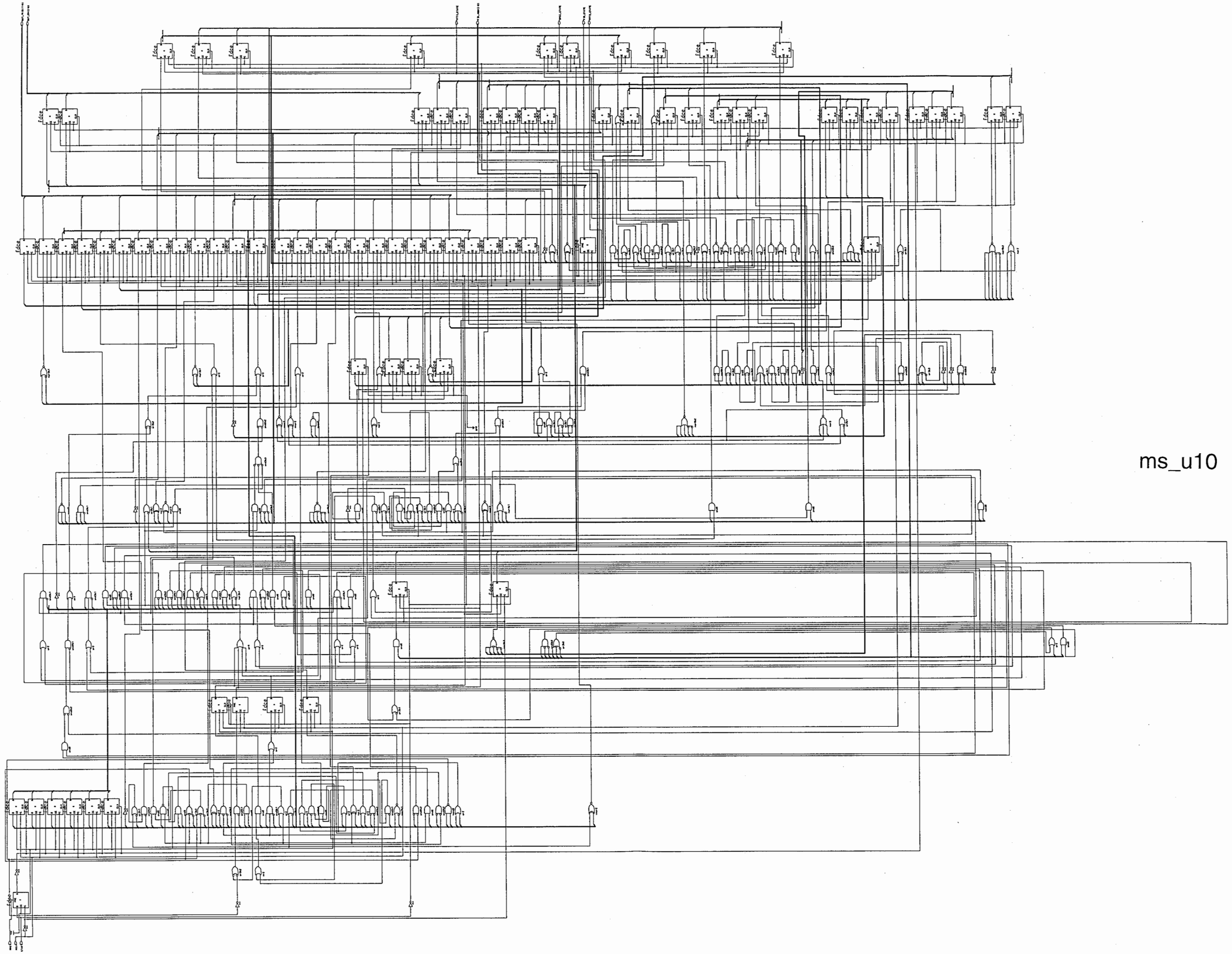


Engineer: taneka	COMPANY NAME
Drawn by: taneka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: mulb_reg
MFG ENGR CHK:	Size: A3
QA CHK:	REV / Drawing Number: / Page: 1

Changed by: taneka | Date Changed: Wednesday, December 13, 1995 | Time Changed: 7:53:24 pm

ss_u1p3





ms_u10

ADD_IN(3:0)

I_WN(7:0)

Q_WN(7:0)

rom16x1	2081	rom16x1	0124
rom16x1	7ADB	rom16x1	5B7E
rom16x1	789B	rom16x1	4B76
rom16x1	22C1	rom16x1	112C
rom16x1	22C1	rom16x1	112C
rom16x1	789B	rom16x1	4B76
rom16x1	22C1	rom16x1	112C
rom16x1	5C1A	rom16x1	CA52

Engineer: taneke
Drawn by: taneke
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:

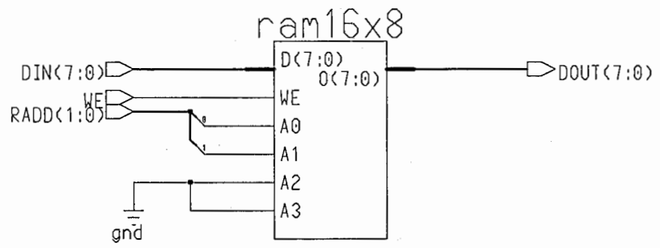
COMPANY NAME
Address
City

TITLE: phc_rom

Size: A3

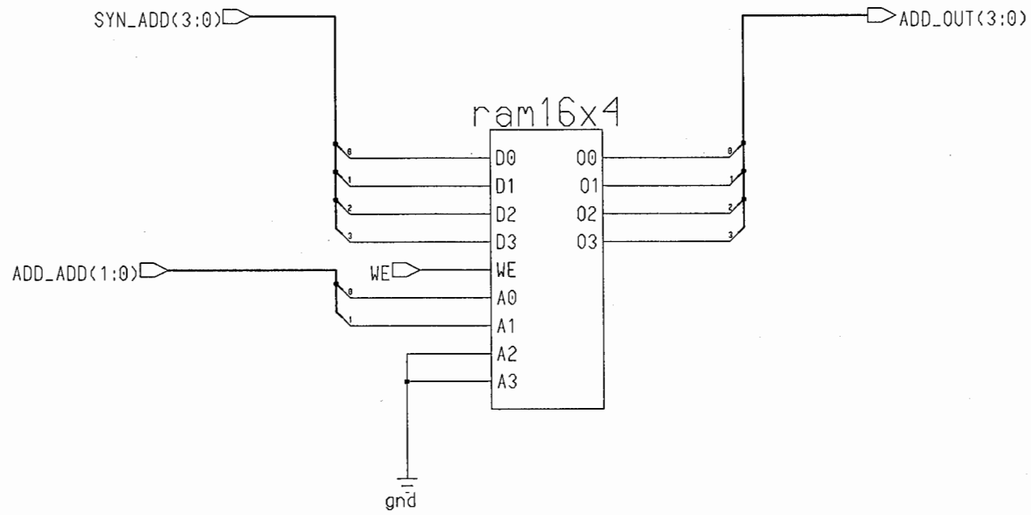
4 1 4 A

Changed by: taneke | Date Changed: Tuesday, October 25, 1994 | Time Changed: 10:13:47 am | QA CHK: | REV | Drawing Number: | Page: 1



Changed by: tanaka	Date Changed: Friday, December 2, 1994	Time Changed: 10:42:41 am														

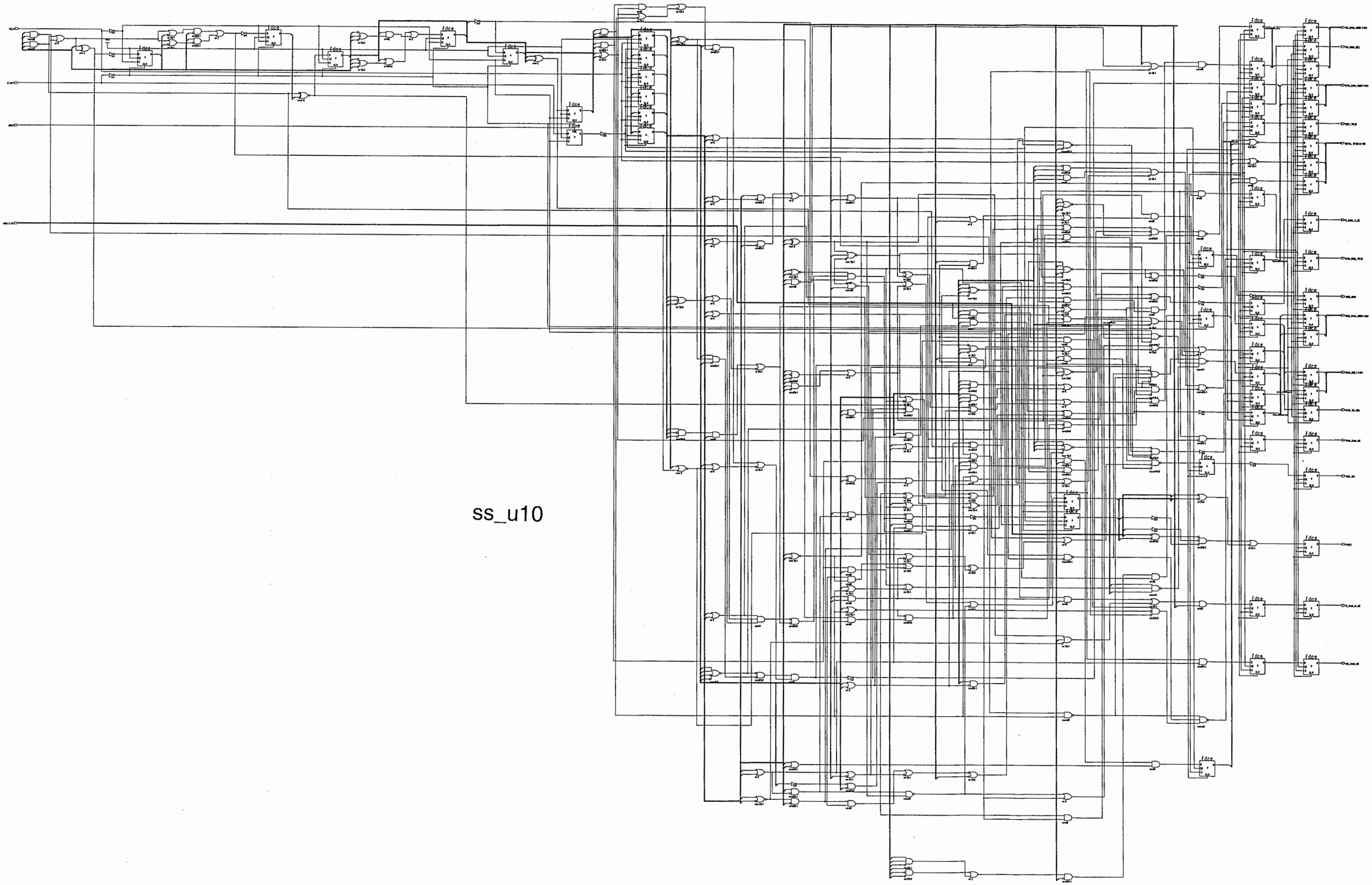
Engineer: tanaka	COMPANY NAME Address City															
Drawn by: tanaka																
R&D CHK:	TITLE: ram48															Size: A3
DOC CTRL CHK:																
MFG ENGR CHK:																
QA CHK:	REV	Drawing Number:														Page: 1

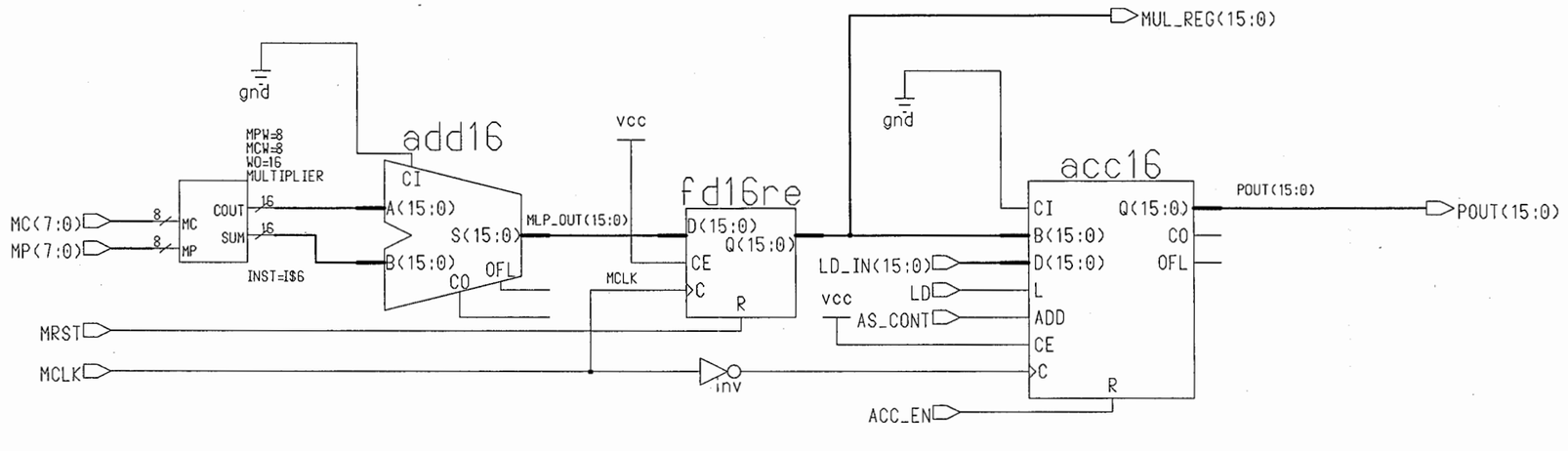


Changed by: taneke	Date Changed: Tuesday, October 25, 1994	Time Changed: 1:42:27 pm	QA CHK:	REV	Drawing Number:	Page: 1

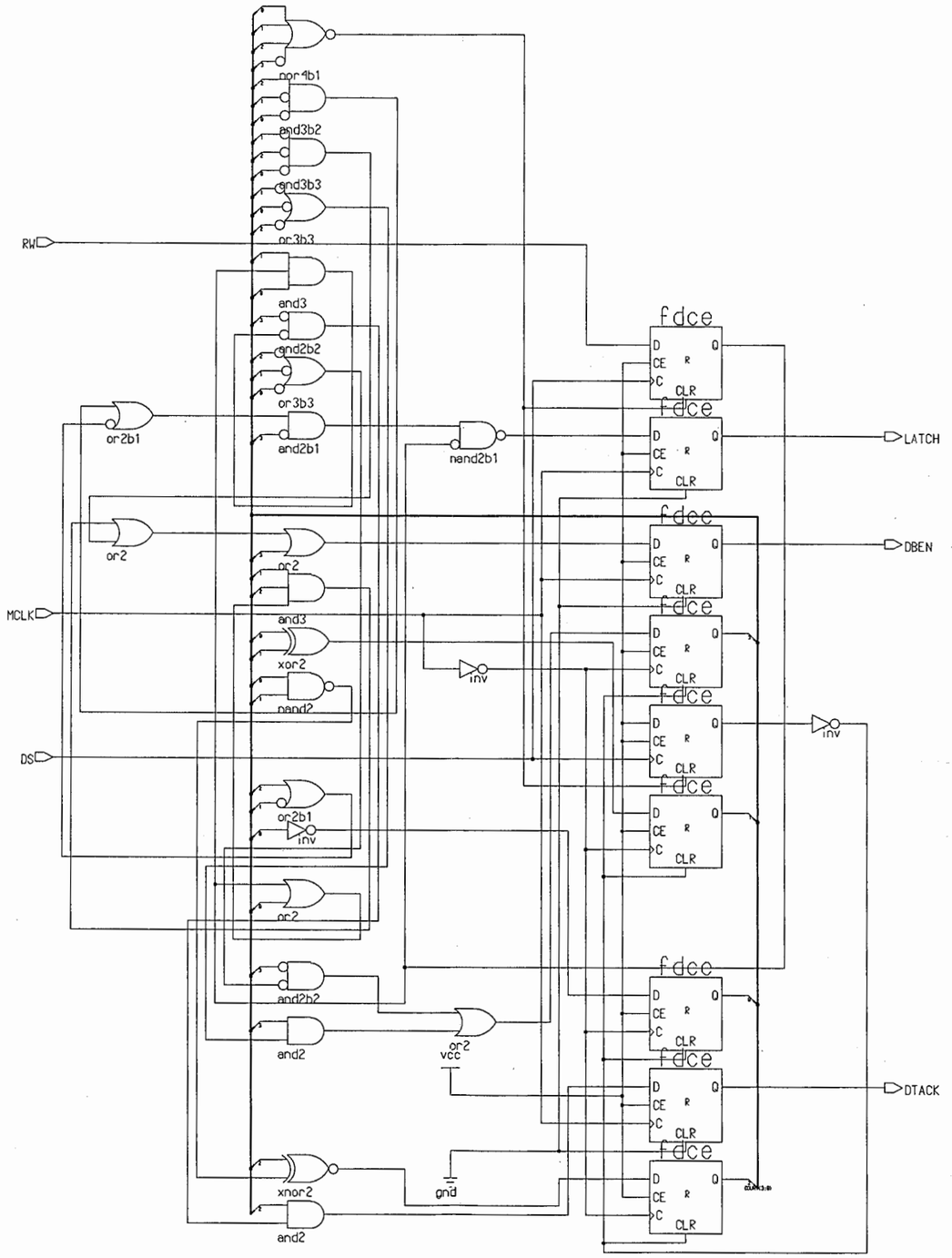
Engineer: taneke	COMPANY NAME Address City	
Drawn by: taneke		
R&D CHK:	TITLE: MC_RAM	Size: A3
DOC CTRL CHK:		4 1 4 A
MFG ENGR CHK:		

ss_u10

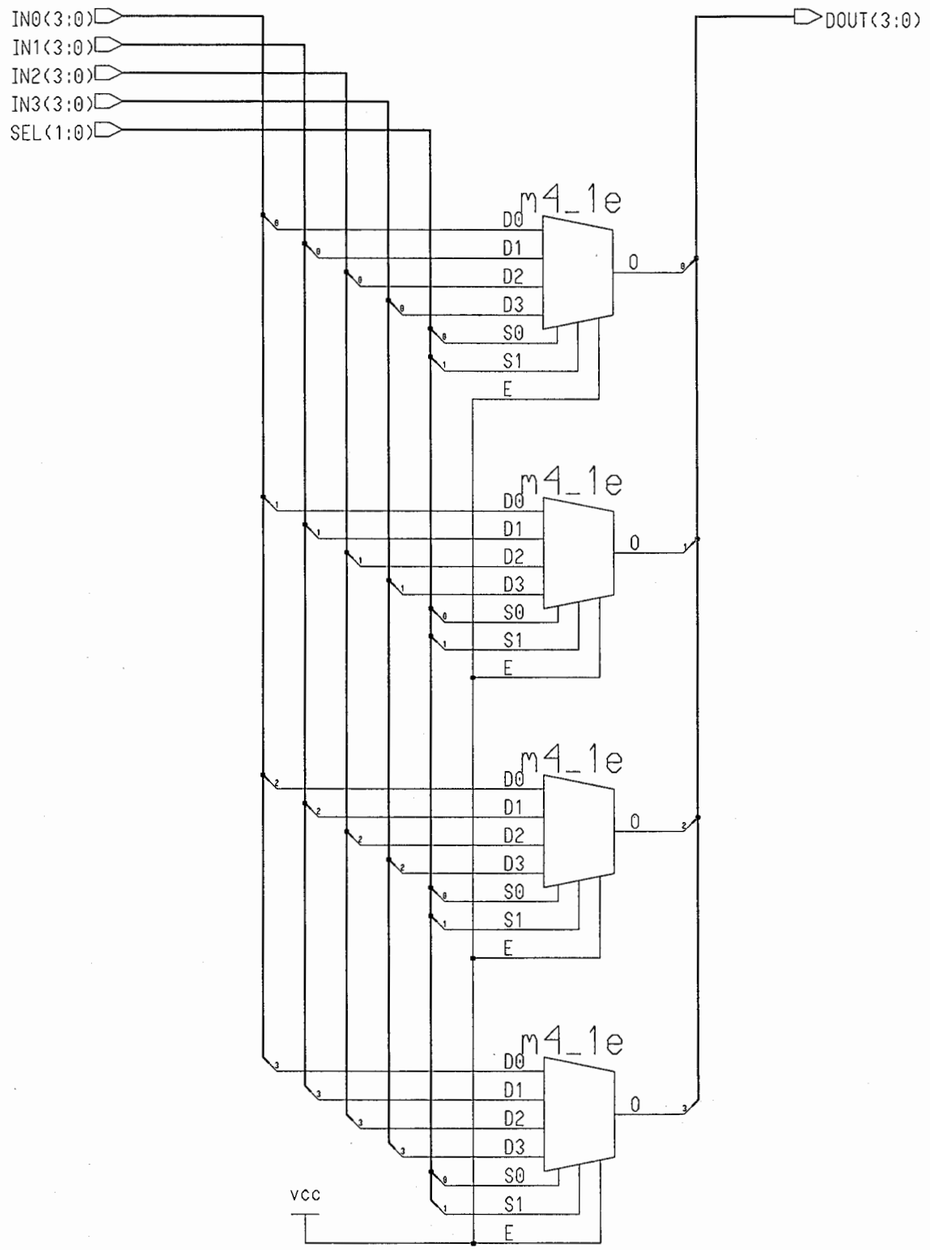




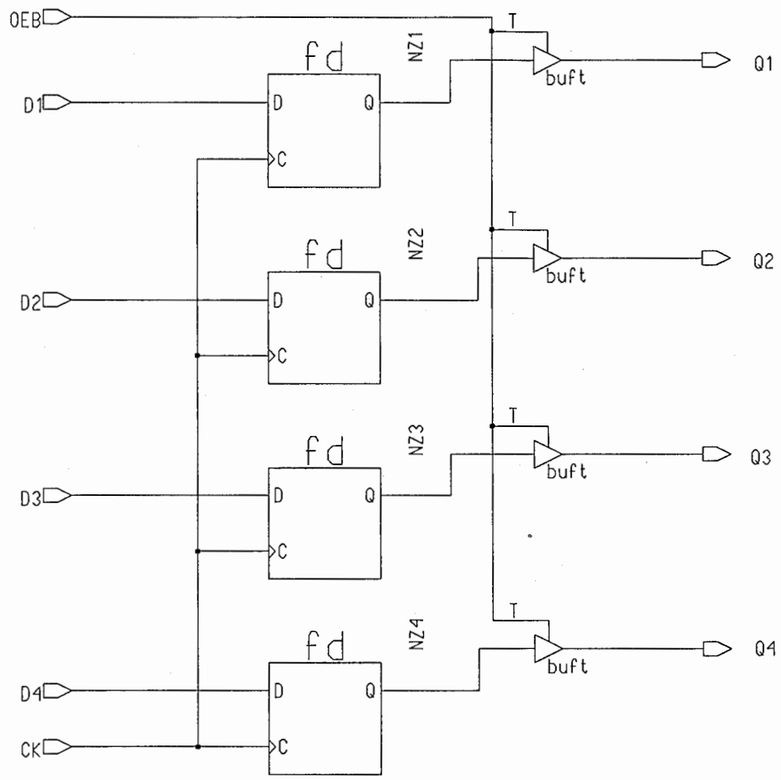
Engineer: tenaka	COMPANY NAME				
Drawn by: tenaka	Address				
R&D CHK:	City				
DOC CTRL CHK:	TITLE: proc8_3			Size: A3	
MFG ENGR CHK:					4 1 4 A
QA CHK:	Changed by: tenaka	Date Changed: Wednesday, August 30, 1995	Time Changed: 7:22:06 pm	REV	Drawing Number: Page: 1



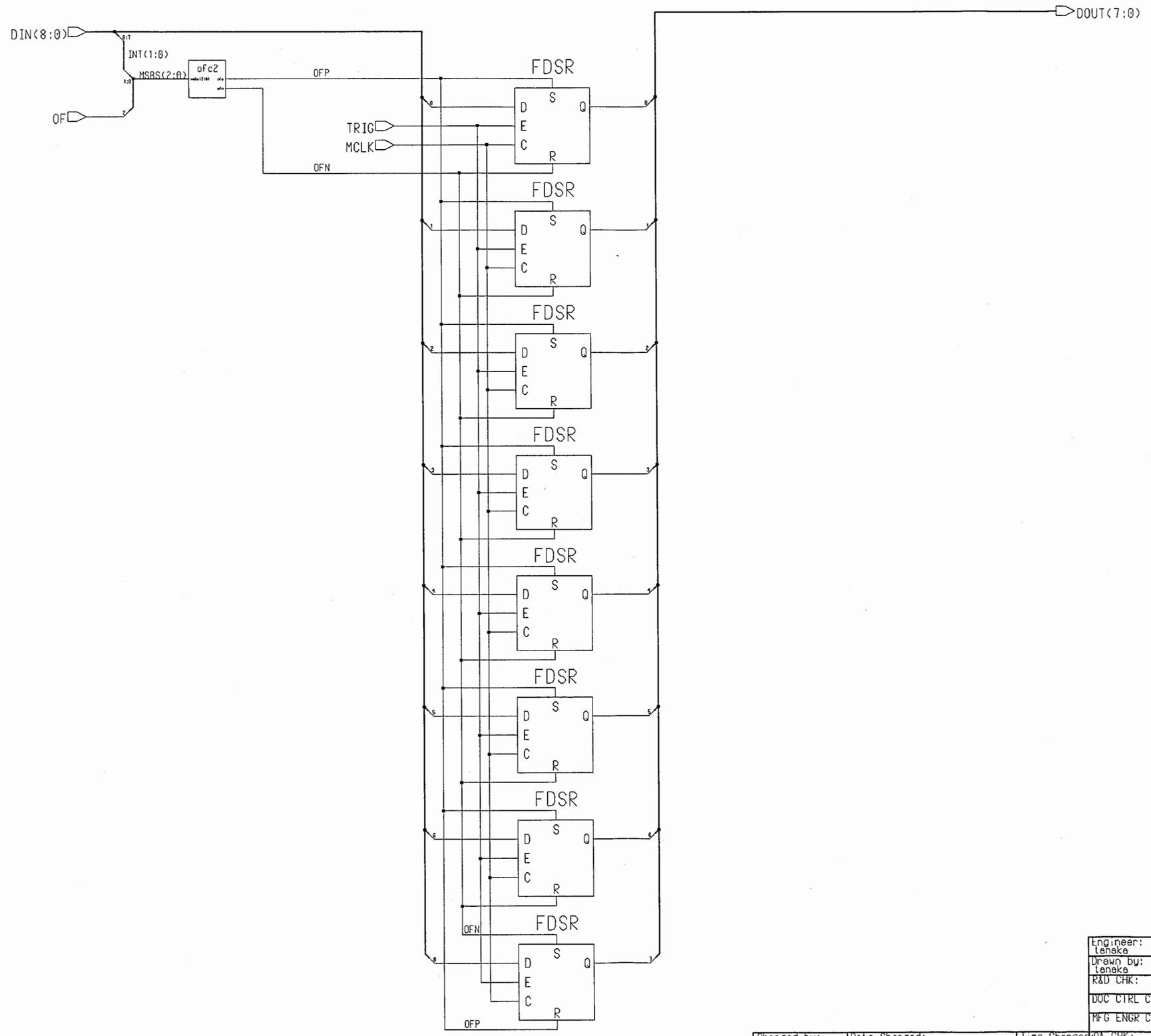
Engineer: tanaka	COMPANY NAME		
Drawn by: tanaka	Address		
R&D CHK:	City		
DOC CTRL CHK:	TITLE: cpu_cont	Size: A2	
MFG ENGR CHK:	QA CHK:	REV	Drawing Number:
Changed by: tanaka	Date Changed: Saturday, March 2, 1996	Time Changed: 11:10:35 am	Page: 1



Engineer: tanaka	COMPANY NAME					
Drawn by: tanaka	Address					
R&D CHK:	City					
DOC CTRL CHK:	TITLE: mux44_1				Size: A3	
MFG ENGR CHK:					4 1 4 A	
Changed by: tanaka	Date Changed: Monday, February 20, 1995	Time Changed: 7:34:56 pm	QA CHK:	REV	Drawing Number:	Page: 1



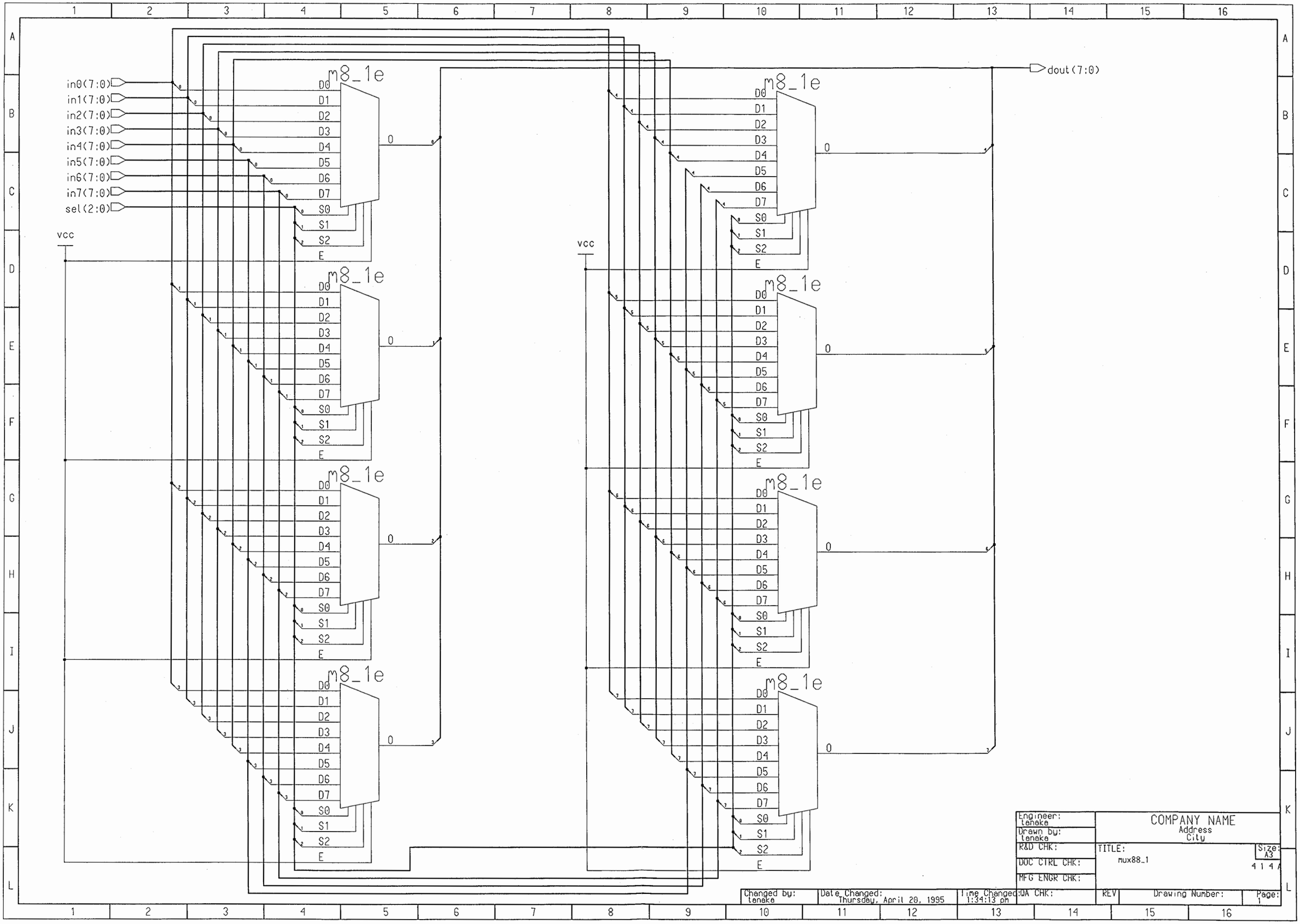
Engineer: taneka	COMPANY NAME
Drawn by: taneka	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: x74_374_2
MFG ENGR CHK:	Size: A3
QA CHK:	REV: 4 1 4 A
Changed by: taneka	Drawing Number:
Date Changed: Friday, September 8, 1995	Page: 1
Time Changed: 11:06:24 am	

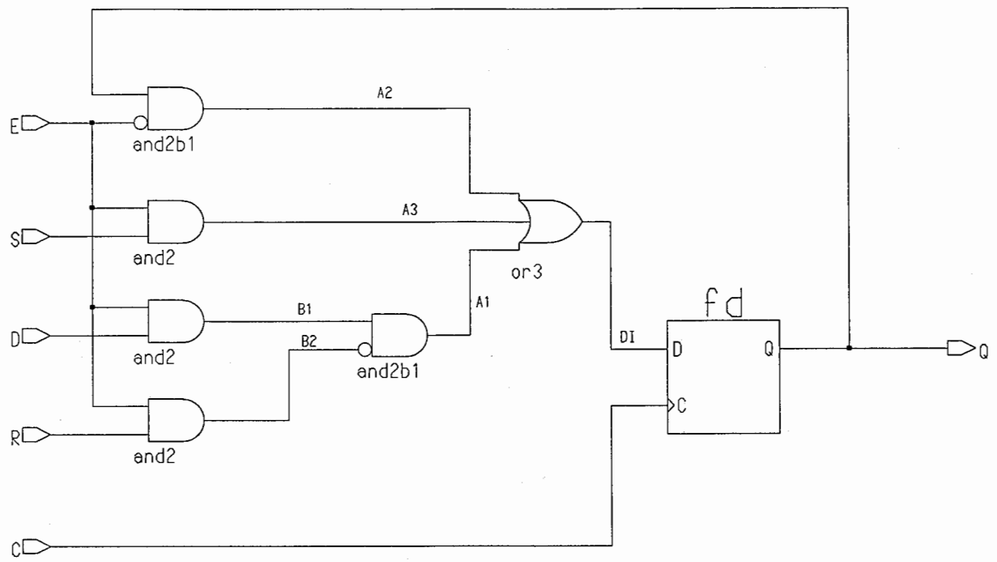


Changed by: taneke	Date Changed: Tuesday, May 2, 1995	Time Changed: 4:22:53 pm	QA CHK:	REV	Drawing Number:

Engineer: taneke	COMPANY NAME
Drawn by: taneke	Address
R&D CHK:	City
DOC CTRL CHK:	TITLE: reg8ofc
MFG ENGR CHK:	Size: A3
	4 1 4 A

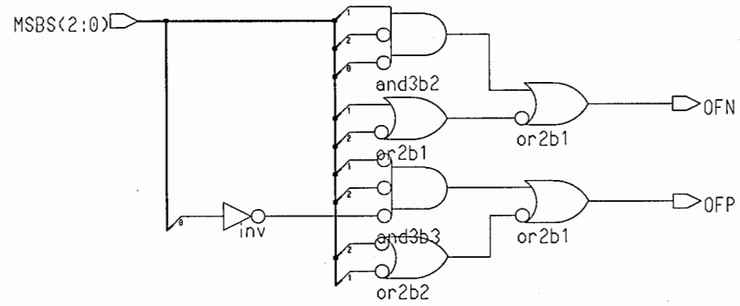
Page: 1


in0(7:0)
in1(7:0)
in2(7:0)
in3(7:0)
in4(7:0)
in5(7:0)
in6(7:0)
in7(7:0)
sel(2:0)
vcc
m8_1e
D0
D1
D2
D3
D4
D5
D6
D7
S0
S1
S2
E
O
dout(7:0)
Engineer: teneke
Drawn by: teneke
R&D CHK:
DOC CTRL CHK:
MFG ENGR CHK:
COMPANY NAME
Address
City
TITLE:
mux88_1
Size: A3
4 1 4 A
Changed by: teneke
Date Changed: Thursday, April 20, 1995
Time Changed: 1:34:13 pm
QA CHK:
REV
Drawing Number:
Page: 1

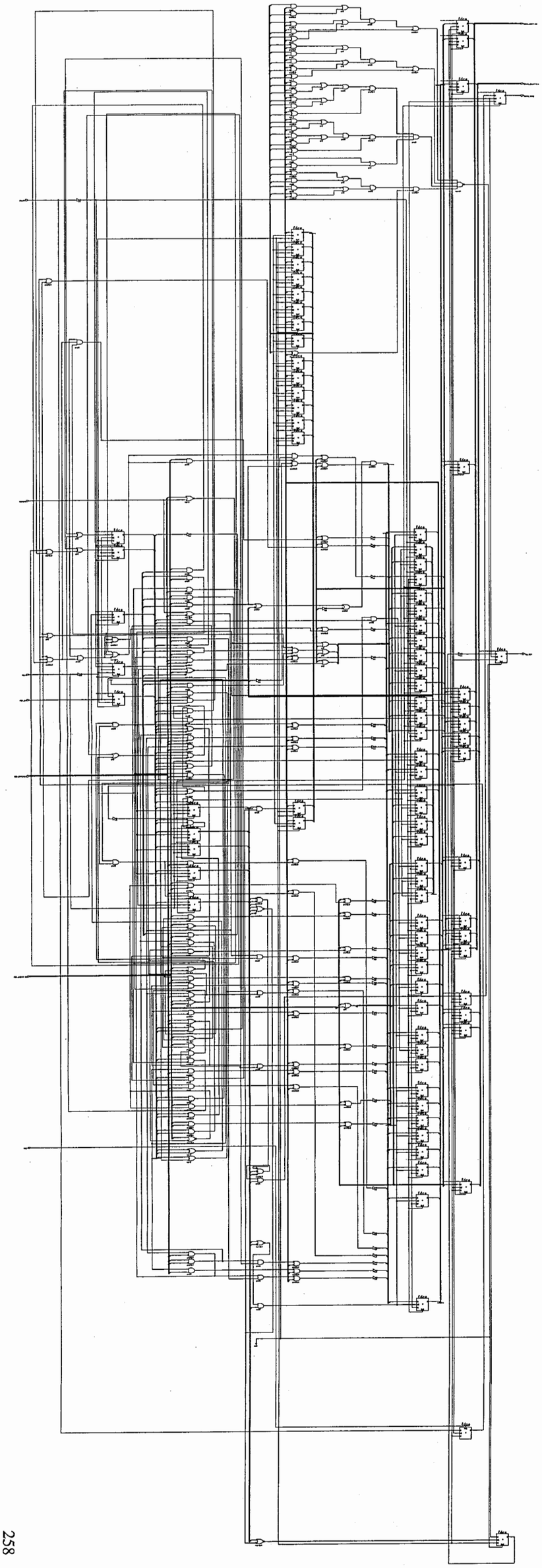


Changed by: taneke	Date Changed: Thursday, April 27, 1995	Time Changed: 1:42:49 pm	QA CHK:	REV	Drawing Number:

Engineer: taneke	COMPANY NAME Address City		
Drawn by: taneke			
R&D CHK:	TITLE: FDRS	Size: A3	Page: 1
DOC CTRL CHK:		4 1 4 A	
MFG ENGR CHK:			



Engineer: tanaka	COMPANY NAME						
Drawn by: tanaka	Address						
R&D CHK:	City						
DOC CTRL CHK:	TITLE: ofc2	Size: A3					
MFG ENGR CHK:		4 1 4 A					
Changed by: tanaka	Date Changed: Friday, March 8, 1996	Time Changed: 8:48:29 pm	QA CHK:	REV	Drawing Number:	Page: 1	



ss_comp4

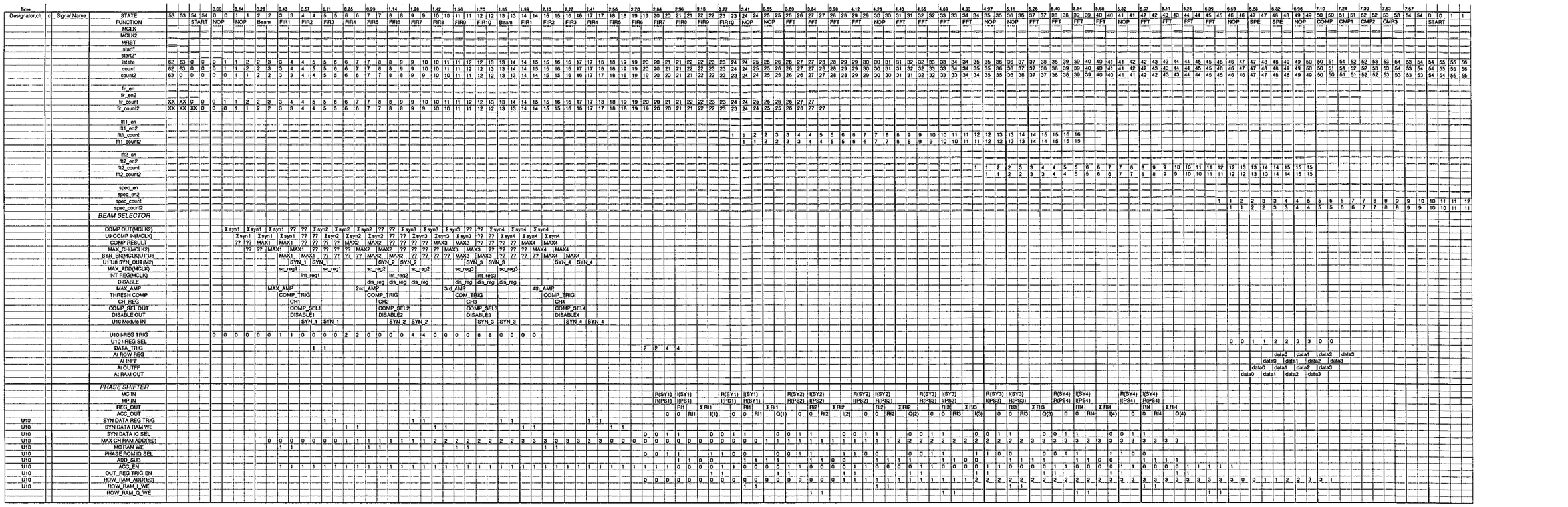
DSP卡一卜FPGA Connection

PIN	Designator	Function	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10
1	A2	I/O(TDI)	TDI	U1-U2	U2-U2	U3-U2	U4-U2	U5-U2	U6-U2	U7-U2	U8-U2	U9-U2
2	A3	I/O	INPUT A(0)	INPUT C(0)	INPUT E(0)	INPUT G(0)	INPUT I(0)	INPUT K(0)	INPUT M(0)	INPUT O(0)	Syn Data OUTR(0)	Syn Data OUTR(0)
3	A4	I/O	INPUT A(1)	INPUT C(1)	INPUT E(1)	INPUT G(1)	INPUT I(1)	INPUT K(1)	INPUT M(1)	INPUT O(1)	Syn Data OUTR(1)	Syn Data OUTR(1)
4	A5	I/O	INPUT A(2)	INPUT C(2)	INPUT E(2)	INPUT G(2)	INPUT I(2)	INPUT K(2)	INPUT M(2)	INPUT O(2)	Syn Data OUTR(2)	Syn Data OUTR(2)
5	A6	I/O	INPUT A(3)	INPUT C(3)	INPUT E(3)	INPUT G(3)	INPUT I(3)	INPUT K(3)	INPUT M(3)	INPUT O(3)	Syn Data OUTR(3)	Syn Data OUTR(3)
6	A7	I/O	INPUT A(4)	INPUT C(4)	INPUT E(4)	INPUT G(4)	INPUT I(4)	INPUT K(4)	INPUT M(4)	INPUT O(4)	Syn Data OUTR(4)	Syn Data OUTR(4)
7	A8	I/O	INPUT A(5)	INPUT C(5)	INPUT E(5)	INPUT G(5)	INPUT I(5)	INPUT K(5)	INPUT M(5)	INPUT O(5)	Syn Data OUTR(5)	Syn Data OUTR(5)
8	A9	I/O	INPUT A(6)	INPUT C(6)	INPUT E(6)	INPUT G(6)	INPUT I(6)	INPUT K(6)	INPUT M(6)	INPUT O(6)	Syn Data OUTR(6)	Syn Data OUTR(6)
9	A10	I/O	INPUT A(7)	INPUT C(7)	INPUT E(7)	INPUT G(7)	INPUT I(7)	INPUT K(7)	INPUT M(7)	INPUT O(7)	Syn Data OUTR(7)	Syn Data OUTR(7)
10	A11	I/O	INPUT A(8)	INPUT C(8)	INPUT E(8)	INPUT G(8)	INPUT I(8)	INPUT K(8)	INPUT M(8)	INPUT O(8)	BAS_FROM_U1	BAS_FROM_U1
11	A12	I/O	INPUT A(9)	INPUT C(9)	INPUT E(9)	INPUT G(9)	INPUT I(9)	INPUT K(9)	INPUT M(9)	INPUT O(9)	DBEN2	DBEN2
12	A13	I/O	INPUT A(10)	INPUT C(10)	INPUT E(10)	INPUT G(10)	INPUT I(10)	INPUT K(10)	INPUT M(10)	INPUT O(10)	CADD	CADD
13	A14	I/O	INPUT A(11)	INPUT C(11)	INPUT E(11)	INPUT G(11)	INPUT I(11)	INPUT K(11)	INPUT M(11)	INPUT O(11)	CADD24	CADD24
14	A15	I/O	INPUT B(0)	INPUT D(0)	INPUT F(0)	INPUT H(0)	INPUT J(0)	INPUT L(0)	INPUT N(0)	INPUT P(0)	Syn Data OUTI(0)	Syn Data OUTI(0)
15	A16	I/O	INPUT B(1)	INPUT D(1)	INPUT F(1)	INPUT H(1)	INPUT J(1)	INPUT L(1)	INPUT N(1)	INPUT P(1)	Syn Data OUTI(1)	Syn Data OUTI(1)
16	A17	I/O	INPUT B(2)	INPUT D(2)	INPUT F(2)	INPUT H(2)	INPUT J(2)	INPUT L(2)	INPUT N(2)	INPUT P(2)	Syn Data OUTI(2)	Syn Data OUTI(2)
17	A18	M0	JM12	JM12	JM12	JM12	JM12	JM12	JM12	JM12	JM12	JM9
18	B1	I/O	INPUT B(3)	INPUT D(3)	INPUT F(3)	INPUT H(3)	INPUT J(3)	INPUT L(3)	INPUT N(3)	INPUT P(3)	Syn Data OUTI(3)	Syn Data OUTI(3)
19	B2	SGCK1(A15,I/O)	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK	PHASE CLK
20	B3	I/O	INPUT B(4)	INPUT D(4)	INPUT F(4)	INPUT H(4)	INPUT J(4)	INPUT L(4)	INPUT N(4)	INPUT P(4)	Syn Data OUTI(4)	Syn Data OUTI(4)
21	B4	I/O(TCK)	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT	OUT_CONT
22	B5	I/O	INPUT B(5)	INPUT D(5)	INPUT F(5)	INPUT H(5)	INPUT J(5)	INPUT L(5)	INPUT N(5)	INPUT P(5)	Syn Data OUTI(5)	Syn Data OUTI(5)
23	B6	I/O	INPUT B(6)	INPUT D(6)	INPUT F(6)	INPUT H(6)	INPUT J(6)	INPUT L(6)	INPUT N(6)	INPUT P(6)	Syn Data OUTI(6)	Syn Data OUTI(6)
24	B7	I/O(TMS)	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL	CMA_OUT_SEL
25	B8	I/O	INPUT B(7)	INPUT D(7)	INPUT F(7)	INPUT H(7)	INPUT J(7)	INPUT L(7)	INPUT N(7)	INPUT P(7)	Syn Data OUTI(7)	Syn Data OUTI(7)
26	B9	I/O	INPUT B(8)	INPUT D(8)	INPUT F(8)	INPUT H(8)	INPUT J(8)	INPUT L(8)	INPUT N(8)	INPUT P(8)	Syn Data OUTI(8)	Syn Data OUTI(8)
27	B10	I/O	INPUT B(9)	INPUT D(9)	INPUT F(9)	INPUT H(9)	INPUT J(9)	INPUT L(9)	INPUT N(9)	INPUT P(9)	Syn Data OUTI(9)	Syn Data OUTI(9)
28	B11	I/O	INPUT B(10)	INPUT D(10)	INPUT F(10)	INPUT H(10)	INPUT J(10)	INPUT L(10)	INPUT N(10)	INPUT P(10)	Syn Data OUTI(10)	Syn Data OUTI(10)
29	B12	I/O	INPUT B(11)	INPUT D(11)	INPUT F(11)	INPUT H(11)	INPUT J(11)	INPUT L(11)	INPUT N(11)	INPUT P(11)	Syn Data OUTI(11)	Syn Data OUTI(11)
30	B13	I/O	AD1	AD1	AD1	AD1	AD1	AD1	AD1	AD1	AD1	MCLK
31	B14	I/O	AD2	AD2	AD2	AD2	AD2	AD2	AD2	AD2	AD2	AD15
32	B15	I/O	AD3	AD3	AD3	AD3	AD3	AD3	AD3	AD3	AD3	AD16
33	B16	SGCK2(I/O)	MRST	MRST	MRST	MRST	MRST	MRST	MRST	MRST	MRST	MRST
34	B17	PGCK2(I/O)	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST	BSYSCLK_DIST
35	B18	I/O	AD4	AD4	AD4	AD4	AD4	AD4	AD4	AD4	AD4	BD12
36	C1	I/O	BR/W*	BR/W*	BR/W*	BR/W*	BR/W*	BR/W*	BR/W*	BR/W*	BR/W*	BSYSFAIL*
37	C2	I/O(A14)	CS1*	CS2*	CS3*	CS4*	CS5*	CS6*	CS7*	CS8*	CS9*	AD14
38	C3	PGCK1(A16,I/O)	MCLK	MCLK	MCLK	MCLK	MCLK	MCLK	MCLK	MCLK	MCLK	MCLK_IN
39	C4	I/O(A17)	BD0	BD0	BD0	BD0	BD0	BD0	BD0	BD0	BD0	BD0
40	C5	I/O	BD1	BD1	BD1	BD1	BD1	BD1	BD1	BD1	BD1	BD1
41	C6	I/O	BD2	BD2	BD2	BD2	BD2	BD2	BD2	BD2	BD2	BD2
42	C7	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
43	C8	I/O	BD3	BD3	BD3	BD3	BD3	BD3	BD3	BD3	BD3	BD3
44	C9	I/O	BD4	BD4	BD4	BD4	BD4	BD4	BD4	BD4	BD4	BD4
45	C10	I/O	BD5	BD5	BD5	BD5	BD5	BD5	BD5	BD5	BD5	BD5
46	C11	I/O	BD6	BD6	BD6	BD6	BD6	BD6	BD6	BD6	BD6	BD6
47	C12	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
48	C13	I/O	BD7	BD7	BD7	BD7	BD7	BD7	BD7	BD7	BD7	BD7
49	C14	I/O	BD8	BD8	BD8	BD8	BD8	BD8	BD8	BD8	BD8	BD8
50	C15	M1	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	JM10
51	C16	M2	JM13	JM13	JM13	JM13	JM13	JM13	JM13	JM13	JM13	JM11
52	C17	I/O	BD9	BD9	BD9	BD9	BD9	BD9	BD9	BD9	BD9	BD9
53	C18	I/O	BD10	BD10	BD10	BD10	BD10	BD10	BD10	BD10	BD10	BD10
54	D1	I/O	BD11	BD11	BD11	BD11	BD11	BD11	BD11	BD11	BD11	BD11
55	D2	I/O(A13)	BD12	BD12	BD12	BD12	BD12	BD12	BD12	BD12	BD12	AD13
56	D3	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
57	D4	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
58	D5	I/O	BD13	BD13	BD13	BD13	BD13	BD13	BD13	BD13	BD13	BD13
59	D6	I/O	BD14	BD14	BD14	BD14	BD14	BD14	BD14	BD14	BD14	BD14
60	D7	I/O	BD15	BD15	BD15	BD15	BD15	BD15	BD15	BD15	BD15	BD15
61	D8	I/O	B23_O(1) 2nd FFT	B23_O(1) 2nd FFT	B23_O(1) 2nd FFT	B23_O(1) 2nd FFT	B24_O(1) 2nd FFT	B24_O(1) 2nd FFT	B24_O(1) 2nd FFT	B24_O(1) 2nd FFT	COMP_IN0(0)	BSERCLK
62	D9	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
63	D10	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
64	D11	I/O	B11_O(0) 1st FFT	B12_O(0) 1st FFT	B13_O(0) 1st FFT	B14_O(0) 1st FFT	B11_O(0) 1st FFT	B12_O(0) 1st FFT	B13_O(0) 1st FFT	B14_O(0) 1st FFT	COMP_IN0(1)	BAS*
65	D12	I/O	B23_O(0) 2nd FFT	B23_O(0) 2nd FFT	B23_O(0) 2nd FFT	B23_O(0) 2nd FFT	B24_O(0) 2nd FFT	B24_O(0) 2nd FFT	B24_O(0) 2nd FFT	B24_O(0) 2nd FFT	COMP_IN0(2)	BDS0*
66	D13	I/O	B11_O(1) 1st FFT	B12_O(1) 1st FFT	B13_O(1) 1st FFT	B14_O(1) 1st FFT	B11_O(1) 1st FFT	B12_O(1) 1st FFT	B13_O(1) 1st FFT	B14_O(1) 1st FFT	COMP_IN0(3)	BDS1*
67	D14	I/O	B11_O(2) 1st FFT	B12_O(2) 1st FFT	B13_O(2) 1st FFT	B14_O(2) 1st FFT	B11_O(2) 1st FFT	B12_O(2) 1st FFT	B13_O(2) 1st FFT	B14_O(2) 1st FFT	COMP_IN0(4)	BDTACK*
68	D15	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
69	D16	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
70	D17	I/O	B11_O(4) 1st FFT	B12_O(4) 1st FFT	B13_O(4) 1st FFT	B14_O(4) 1st FFT	B11_O(4) 1st FFT	B12_O(4) 1st FFT	B13_O(4) 1st FFT	B14_O(4) 1st FFT	COMP_IN0(5)	BRW*
71	D18	I/O	B11_O(7) 1st FFT	B12_O(7) 1st FFT	B13_O(7) 1st FFT	B14_O(7) 1st FFT	B11_O(7) 1st FFT	B12_O(7) 1st FFT	B13_O(7) 1st FFT	B14_O(7) 1st FFT	COMP_IN0(6)	Syn_OUT_EN(0)
72	E1	I/O	B23_O(2) 2nd FFT	B23_O(2) 2nd FFT	B23_O(2) 2nd FFT	B23_O(2) 2nd FFT	B24_O(2) 2nd FFT	B24_O(2) 2nd FFT	B24_O(2) 2nd FFT	B24_O(2) 2nd FFT	COMP_IN0(7)	Syn_OUT_EN(1)
73	E2	I/O	B11_I(4) 1st FFT	B12_I(4) 1st FFT	B13_I(4) 1st FFT	B14_I(4) 1st FFT	B11_I(4) 1st FFT	B12_I(4) 1st FFT	B13_I(4) 1st FFT	B14_I(4) 1st FFT	COMP_IN0(8)	Syn_OUT_EN(2)
74	E3	I/O	B11_I(0) 1st FFT	B12_I(0) 1st FFT	B13_I(0) 1st FFT	B14_I(0) 1st FFT	B11_I(0) 1st FFT	B12_I(0) 1st FFT	B13_I(0) 1st FFT	B14_I(0) 1st FFT	COMP_IN0(9)	Syn_OUT_EN(3)
75	E4	I/O	B11_I(1) 1st FFT	B12_I(1) 1st FFT	B13_I(1) 1st FFT	B14_I(1) 1st FFT	B11_I(1) 1st FFT	B12_I(1) 1st FFT	B13_I(1) 1st FFT	B14_I(1) 1st FFT	COMP_IN1(0)	BSERDATA
76	E15	I/O	B21_I(1) 2nd FFT	B21_I(1) 2nd FFT	B21_I(1) 2nd FFT	B21_I(1) 2nd FFT	B22_I(1) 2nd FFT	B22_I(1) 2nd FFT	B22_I(1) 2nd FFT	B22_I(1) 2nd FFT	COMP_IN1(1)	BD16
77	E16	I/O(HDC)	B11_O(3) 1st FFT	B12_O(3) 1st FFT	B13_O(3) 1st FFT	B14_O(3) 1st FFT	B11_O(3) 1st FFT	B12_O(3) 1st FFT	B13_O(3) 1st FFT	B14_O(3) 1st FFT	COMP_IN1(2)	BD17
78	E17	I/O(LDC*)	B11_O(5) 1st FFT	B12_O(5) 1st FFT	B13_O(5) 1st FFT	B14_O(5) 1st FFT	B11_O(5) 1st FFT	B12_O(5) 1st FFT	B13_O(5) 1st FFT	B14_O(5) 1st FFT	COMP_IN1(3)	BD18
79	E18	I/O	AUX4	AUX4	AUX4	AUX4	AUX4	AUX4	AUX4	AUX4	DSP_OUT(8)	ADC CLK OUT(J3,J8)
80	F1	I/O	LATCH	LATCH	LATCH	LATCH	LATCH	LATCH	LATCH	LATCH	LATCH	LATCH
81	F2	I/O	B11_I(5) 1st FFT	B12_I(5) 1st FFT	B13_I(5) 1st FFT	B14_I(5) 1st FFT	B11_I(5) 1st FFT	B12_I(5) 1st FFT	B13_I(5) 1st FFT	B14_I(5) 1st FFT	COMP_IN1(4)	BD19
82	F3	I/O(A12)	B11_I(3) 1st FFT	B12_I(3) 1st FFT	B13_I(3) 1st FFT	B14_I(3) 1st FFT	B11_I(3) 1st FFT	B12_I(3) 1st FFT	B13_I(3) 1st FFT	B14_I(3) 1st FFT	COMP_IN1(5)	AD12
83	F4	I/O	B11_I(2) 1st FFT	B12_I(2) 1st FFT	B13_I(2) 1st FFT	B14_I(2) 1st FFT	B11_I(2) 1st FFT	B12_I(2) 1st FFT	B13_I(2) 1st FFT	B14_I(2) 1st FFT	COMP_IN1(6)	A/D B'd enable*
84	F15	I/O	B21_I(2) 2nd FFT	B21_I(2) 2nd FFT	B21_I(2) 2nd FFT	B21_I(2) 2nd FFT	B22_I(2) 2nd FFT	B22_I(2) 2nd FFT	B22_I(2) 2nd FFT	B22_I(2) 2nd FFT	COMP_IN1(7)	BUSY* From AC
85	F16	I/O	B11_Q(6) 1st FFT	B12_O(6) 1st FFT	B13_O(6) 1st FFT	B14_O(6) 1st FFT	B11_Q(6) 1st FFT	B12_O(6) 1st FFT	B13_O(6) 1st FFT	B14_O(6) 1st FFT	COMP_IN1(8)	LED10
86	F17	I/O	B21_I(0) 1st FFT	B21_I(0) 1st FFT	B21_I(0) 1st FFT	B21_I(0) 1st FFT	B22_I(0) 1st FFT	B22_I(0) 1st FFT	B22_I(0) 1st FFT	B22_I(0) 1st FFT	COMP_IN1(9)	BMODID
87	F18	I/O	B21_I(3) 1st FFT	B21_I(3) 1st FFT	B21_I(3) 1st FFT	B21_I(3) 1st FFT	B22_I(3) 1st FFT	B22_I(3) 1st FFT	B22_I(3) 1st FFT	B22_I(3) 1st FFT	COMP_IN2(0)	BLWORD*
88	G1	I/O(A10)	B11_I(7) 1st FFT	B12_I(7) 1st FFT	B13_I(7) 1st FFT	B14_I(7) 1st FFT	B11_I(7) 1st FFT	B12_I(7) 1st FFT	B13_I(7) 1st FFT	B14_I(7) 1st FFT	COMP_IN2(1)	AD10
89	G2	I/O(A11)	B11_I(6) 1st FFT	B12_I(6) 1st FFT	B13_I(6) 1st FFT	B14_I(6) 1st FFT	B11_I(6) 1st FFT	B12_I(6) 1st FFT	B13_I(6) 1st FFT	B14_I(6) 1st FFT	COMP_IN2(2)	AD11
90	G3	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
91	G4	I/O	B23_O(3) 1st FFT	B23_O(3) 1st FFT	B23_O(3) 1st FFT	B23_O(3) 1st FFT	B24_O(3) 1st FFT	B24_O(3) 1st FFT	B24_O(3) 1st FFT	B24_O(3) 1st FFT	COMP_IN2(3)	DSP_OUT(0)
92	G15	I/O	B21_O(0) 2nd FFT	B21_O(0) 2nd FFT	B21_O(0) 2nd FFT	B21_O(0) 2nd FFT	B22_O(0) 2nd FFT	B22_O(0) 2nd FFT	B22_O(0) 2nd FFT	B22_O(0) 2nd FFT	COMP_IN2(4)	DSP_OUT(1)
93	G16	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
94	G17	I/O	B21_I(4) 2nd FFT	B21_I(4) 2nd FFT	B21_I(4) 2nd FFT	B21_I(4) 2nd FFT	B22_I(4) 2nd FFT	B22_I(4) 2nd FFT	B22_I(4) 2nd FFT	B22_I(4) 2nd FFT	COMP_IN2(5)	DSP_OUT(2)
95	G18	I/O	B21_I(5) 2nd FFT	B21_I(5) 2nd FFT	B21_I(5) 2nd FFT	B21_I(5) 2nd FFT	B22_I(5) 2nd FFT	B22_I(5) 2nd FFT	B22_I(5) 2nd FFT	B22_I(5) 2nd FFT	COMP_IN2(6)	DSP_OUT(3)
96	H1	I/O	B23_O(7) 2nd FFT	B23_O(7) 2nd FFT	B23_O(7) 2nd FFT	B23_O(7) 2nd FFT	B24_O(7) 2nd FFT	B24_O(7) 2nd FFT	B24_O(7) 2nd FFT	B24_O(7) 2nd FFT	COMP_IN2(7)	DSP_OUT(4)
97	H2	I/O	B23_O(6) 2nd FFT	B23_O(6) 2nd FFT	B23_O(6) 2nd FFT	B23_O(6) 2nd FFT	B24_O(6) 2nd FFT	B24_O(6) 2nd FFT	B24_O(6) 2nd FFT	B24_O(6) 2nd FFT	COMP_IN2(8)	DSP_OUT(5)
98	H3	I/O	B23_O(5) 2nd FFT	B23_O(5) 2nd FFT	B23_O(5) 2nd FFT	B23_O(5) 2nd FFT	B24_O(5) 2nd FFT	B24_O(5) 2nd FFT	B24_O(5) 2nd FFT	B24_O(5) 2nd FFT	COMP_IN2(9)	DSP_OUT(6)
99	H4	I/O	B23_O(4) 2nd FFT	B23_O(4) 2nd FFT	B23_O(4) 2nd FFT	B23_O(4) 2nd FFT	B24_O(4) 2nd FFT	B24_O(4) 2nd FFT	B24_O(4) 2nd FFT	B24_O(4) 2nd FFT	COMP_IN3(0)	DSP_OUT(7)
100	H15	I/O	B21_O(1) 2nd FFT	B21_O(1) 2nd FFT	B21_O(1) 2nd FFT	B21_O(1) 2nd FFT	B22_O(1) 2nd FFT	B22_O(1) 2nd FFT	B22_O(1) 2nd FFT	B22_O(1) 2nd FFT	COMP_IN3(1)	DSP_OUT(8)
101	H16	I/O	B21_I(6) 2nd FFT	B21_I(6) 2nd FFT	B21_I(6) 2nd FFT	B21_I(6) 2nd FFT	B22_I(6) 2nd FFT	B22_I(6) 2nd FFT	B22_I(6) 2nd FFT	B22_I(6) 2nd FFT	COMP_IN3(2)	DSP_OUT(9)
102	H17	I/O	B21_I(7) 2nd FFT	B21_I(7) 2nd FFT	B21_I(7) 2nd FFT	B21_I(7) 2nd FFT	B22_I(7) 2nd FFT	B22_I(7) 2nd FFT	B22_I(7) 2nd FFT	B22_I(7) 2nd FFT	COMP_IN3(3)	DSP_OUT(10)
103	H18	I/O	B21_O(2) 2nd FFT	B21_O(2) 2nd FFT	B21_O(2) 2nd FFT	B21_O(2) 2nd FFT	B22_O(2) 2nd FFT	B22_O(2) 2nd FFT	B22_O(2) 2nd FFT	B22_O(2) 2nd FFT	COMP_IN3(4)	DSP_OUT(11)
104	J1	I/O	B23_I(0) 2nd FFT	B23_I(0) 2nd FFT	B23_I(0) 2nd FFT	B23_I(0) 2nd FFT	B24_I(0) 2nd FFT	B24_I(0) 2nd FFT	B24_I(0) 2nd FFT	B24_I(0) 2nd FFT	COMP_IN3(5)	DSP_OUT(12)
105	J2	I/O(A9)	B23_I(1) 2nd FFT	B23_I(1) 2nd FFT	B23_I(1) 2nd FFT	B23_I(1) 2nd FFT	B24_I(1) 2nd FFT	B24_I(1) 2nd FFT	B24_I(1) 2nd FFT	B24_I(1) 2nd FFT	COMP_IN3(6)	AD9
106	J3	I/O(A8)	B23_I(2) 2nd FFT	B23_I(2) 2nd FFT	B23_I(2) 2nd FFT	B23_I(2) 2nd FFT	B24_I(2) 2nd FFT	B24_I(2) 2nd FFT	B24_I(2) 2nd FFT	B24_I(2) 2nd FFT	COMP_IN3(7)	AD8
107	J4	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
108	J15	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
109	J16	I/O(ERR*,INIT*)	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT	SNAP_SHOT
110	J17	I/O	B21_O(4) 2nd FFT	B21_O(4) 2nd FFT	B21_O(4) 2nd FFT	B21_O(4) 2nd FFT	B22_O(4) 2nd FFT	B22_O(4) 2nd FFT	B22_O(4) 2nd FFT	B22_O(4) 2nd FFT	COMP_IN3(8)	DSP_OUT(13)
111	J18	I/O	B21_O(3) 2nd FFT	B21_O(3) 2nd FFT	B21_O(3) 2nd FFT	B21_O(3) 2nd FFT	B22_O(3) 2nd FFT	B22_O(3) 2nd FFT	B22_O(3) 2nd FFT	B22_O(3) 2nd FFT	COMP_IN3(9)	DSP_OUT(14)
112	K1	I/O	B23_I(5) 2nd FFT	B23_I(5) 2nd FFT	B23_I(5) 2nd FFT	B23_I(5) 2nd FFT	B24_I(5) 2nd FFT	B24_I(5) 2nd FFT	B24_I(5) 2nd FFT	B24_I(5) 2nd FFT	COMP_IN4(0)	DSP_OUT(15)
113	K2	I/O(A5)	B23_I(4) 2nd FFT	B23_I(4) 2nd FFT	B23_I(4) 2nd FFT	B23_I(4) 2nd FFT	B24_I(4) 2nd FFT	B24_I(4) 2nd FFT	B24_I(4) 2nd FFT	B24_I(4) 2nd FFT	COMP_IN4(1)	AD6

114	K3	I/O(A7)	B23_I(3) 2nd FFT	B23_I(3) 2nd FFT	B23_I(3) 2nd FFT	B23_I(3) 2nd FFT	B24_I(3) 2nd FFT	B24_I(3) 2nd FFT	B24_I(3) 2nd FFT	B24_I(3) 2nd FFT	COMP_IN4(2)	AD7
115	K4	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
116	K15	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
117	K16	I/O	B21_O(5) 2nd FFT	B21_O(5) 2nd FFT	B21_O(5) 2nd FFT	B21_O(5) 2nd FFT	B22_O(5) 2nd FFT	B22_O(5) 2nd FFT	B22_O(5) 2nd FFT	B22_O(5) 2nd FFT	B22_O(5) 2nd FFT	DSP_OUT(16)
118	K17	I/O	B21_O(6) 2nd FFT	B21_O(6) 2nd FFT	B21_O(6) 2nd FFT	B21_O(6) 2nd FFT	B22_O(6) 2nd FFT	B22_O(6) 2nd FFT	B22_O(6) 2nd FFT	B22_O(6) 2nd FFT	B22_O(6) 2nd FFT	DSP_OUT(17)
119	K18	I/O	B21_Q(7) 2nd FFT	B21_Q(7) 2nd FFT	B21_Q(7) 2nd FFT	B21_Q(7) 2nd FFT	B22_Q(7) 2nd FFT	B22_Q(7) 2nd FFT	B22_Q(7) 2nd FFT	B22_Q(7) 2nd FFT	B22_Q(7) 2nd FFT	DSP_OUT(18)
120	L1	I/O	B23_I(6) 2nd FFT	B23_I(6) 2nd FFT	B23_I(6) 2nd FFT	B23_I(6) 2nd FFT	B24_I(6) 2nd FFT	B24_I(6) 2nd FFT	B24_I(6) 2nd FFT	B24_I(6) 2nd FFT	B24_I(6) 2nd FFT	DSP_OUT(19)
121	L2	I/O	B23_I(7) 2nd FFT	B23_I(7) 2nd FFT	B23_I(7) 2nd FFT	B23_I(7) 2nd FFT	B24_I(7) 2nd FFT	B24_I(7) 2nd FFT	B24_I(7) 2nd FFT	B24_I(7) 2nd FFT	B24_I(7) 2nd FFT	DSP_OUT(20)
122	L3	I/O	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	Syn Data OUTR(0)	DSP_OUT(21)
123	L4	I/O	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	Syn Data OUTR(1)	DSP_OUT(22)
124	L15	I/O	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	Syn Data OUTR(2)	COMP_IN5(0)	DSP_OUT(23)
125	L16	I/O	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	Syn Data OUTR(3)	COMP_IN5(1)	AD18
126	L17	I/O	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	Syn Data OUTR(4)	COMP_IN5(2)	AD19
127	L18	I/O	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	Syn Data OUTR(5)	COMP_IN5(3)	AD20
128	M1	I/O(A5)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	Syn Data OUTR(6)	COMP_IN5(4)	AD5
129	M2	I/O(A4)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	Syn Data OUTR(7)	COMP_IN5(5)	AD4
130	M3	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
131	M4	I/O	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	Syn Data OUTR(8)	COMP_IN5(6)	AD21
132	M15	I/O	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	Syn Data OUTR(9)	COMP_IN5(7)	AD22
133	M16	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
134	M17	I/O	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	Syn Data OUTR(10)	COMP_IN5(8)	AD23
135	M18	I/O	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	Syn Data OUTR(11)	COMP_IN5(9)	BD20
136	N1	I/O	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	Syn Data OUTI(0)	COMP_IN6(0)	BD21
137	N2	I/O	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	Syn Data OUTI(1)	COMP_IN6(1)	BD22
138	N3	I/O(A3)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	Syn Data OUTI(2)	COMP_IN6(2)	AD3
139	N4	I/O	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	Syn Data OUTI(3)	COMP_IN6(3)	BD23
140	N15	I/O	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	Syn Data OUTI(4)	COMP_IN6(4)	BD24
141	N16	I/O	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	Syn Data OUTI(5)	COMP_IN6(5)	BD25
142	N17	I/O	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	Syn Data OUTI(6)	COMP_IN6(6)	BD26
143	N18	I/O	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	Syn Data OUTI(7)	COMP_IN6(7)	BD27
144	P1	I/O	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	BAS_FROM_U1	COMP_IN6(8)	BD28
145	P2	I/O	DBEN2	DBEN2	DBEN2	DBEN2	DBEN2	DBEN2	DBEN2	DBEN2	COMP_IN6(9)	BD29
146	P3	I/O	CADD	CADD	CADD	CADD	CADD	CADD	CADD	CADD	COMP_IN7(0)	BD30
147	P4	I/O	CADD24	CADD24	CADD24	CADD24	CADD24	CADD24	CADD24	CADD24	COMP_IN7(1)	BD31
148	P15	I/O	COMP_IN0(0)	COMP_IN1(0)	COMP_IN2(0)	COMP_IN3(0)	COMP_IN4(0)	COMP_IN5(0)	COMP_IN6(0)	COMP_IN7(0)	COMP_IN7(2)	BAM0
149	P16	I/O	COMP_IN0(1)	COMP_IN1(1)	COMP_IN2(1)	COMP_IN3(1)	COMP_IN4(1)	COMP_IN5(1)	COMP_IN6(1)	COMP_IN7(1)	COMP_IN7(3)	BAM1
150	P17	I/O	COMP_IN0(2)	COMP_IN1(2)	COMP_IN2(2)	COMP_IN3(2)	COMP_IN4(2)	COMP_IN5(2)	COMP_IN6(2)	COMP_IN7(2)	COMP_IN7(4)	BAM2
151	P18	I/O	COMP_IN0(3)	COMP_IN1(3)	COMP_IN2(3)	COMP_IN3(3)	COMP_IN4(3)	COMP_IN5(3)	COMP_IN6(3)	COMP_IN7(3)	COMP_IN7(5)	BAM3
152	R1	I/O	COMP_IN0(4)	COMP_IN1(4)	COMP_IN2(4)	COMP_IN3(4)	COMP_IN4(4)	COMP_IN5(4)	COMP_IN6(4)	COMP_IN7(4)	COMP_IN7(6)	BAM4
153	R2	I/O	COMP_IN0(5)	COMP_IN1(5)	COMP_IN2(5)	COMP_IN3(5)	COMP_IN4(5)	COMP_IN5(5)	COMP_IN6(5)	COMP_IN7(5)	COMP_IN7(7)	BAM5
154	R3	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
155	R4	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
156	R5	I/O	COMP_IN0(6)	COMP_IN1(6)	COMP_IN2(6)	COMP_IN3(6)	COMP_IN4(6)	COMP_IN5(6)	COMP_IN6(6)	COMP_IN7(6)	COMP_IN7(8)	JM12(M0_OUT)
157	R6	I/O	COMP_IN0(7)	COMP_IN1(7)	COMP_IN2(7)	COMP_IN3(7)	COMP_IN4(7)	COMP_IN5(7)	COMP_IN6(7)	COMP_IN7(7)	COMP_IN7(9)	BIACKIN*
158	R7	I/O	COMP_IN0(8)	COMP_IN1(8)	COMP_IN2(8)	COMP_IN3(8)	COMP_IN4(8)	COMP_IN5(8)	COMP_IN6(8)	COMP_IN7(8)	ADD_IN(0)	BIACKOUT*
159	R8	I/O	COMP_IN0(9)	COMP_IN1(9)	COMP_IN2(9)	COMP_IN3(9)	COMP_IN4(9)	COMP_IN5(9)	COMP_IN6(9)	COMP_IN7(9)	ADD_IN(1)	BIACK*
160	R9	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
161	R10	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
162	R11	I/O	ADD_IN(0)	ADD_IN(1)	ADD_IN(2)	ADD_IN(3)	ADD_IN(4)	ADD_IN(5)	ADD_IN(6)	ADD_IN(7)	ADD_IN(2)	Filter Data(0)
163	R12	I/O	AUX0	AUX0	AUX0	AUX0	AUX0	AUX0	AUX0	AUX0	ADD_IN(3)	Filter Data(1)
164	R13	I/O	Filter Data(0)	Filter Data(0)	Filter Data(0)	Filter Data(0)	Filter Data(0)	Filter Data(0)	Filter Data(0)	Filter Data(0)	ADD_IN(4)	Filter Data(2)
165	R14	I/O	Filter Data(1)	Filter Data(1)	Filter Data(1)	Filter Data(1)	Filter Data(1)	Filter Data(1)	Filter Data(1)	Filter Data(1)	ADD_IN(5)	Filter Data(3)
166	R15	VCC	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
167	R16	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
168	R17	I/O	Filter Data(2)	Filter Data(2)	Filter Data(2)	Filter Data(2)	Filter Data(2)	Filter Data(2)	Filter Data(2)	Filter Data(2)	ADD_IN(6)	Filter Data(4)
169	R18	I/O	Filter Data(3)	Filter Data(3)	Filter Data(3)	Filter Data(3)	Filter Data(3)	Filter Data(3)	Filter Data(3)	Filter Data(3)	ADD_IN(7)	Filter Data(5)
170	T1	I/O	Filter Data(4)	Filter Data(4)	Filter Data(4)	Filter Data(4)	Filter Data(4)	Filter Data(4)	Filter Data(4)	Filter Data(4)	Syn_OUT_EN(0)	Filter Data(6)
171	T2	I/O(CS1,A2)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Filter Data(5)	Syn_OUT_EN(1)	AD2
172	T3	I/O(A0,WS*)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Filter Data(6)	Syn_OUT_EN(2)	JM13(M2_OUT)
173	T4	SGCK4(DOUT,VO)	MCLK_OUT	U1-U3	U2-U3	U3-U3	U4-U3	U5-U3	U6-U3	U7-U3	U8-U3	THRESH
174	T5	I/O	Filter Data(7)	Filter Data(7)	Filter Data(7)	Filter Data(7)	Filter Data(7)	Filter Data(7)	Filter Data(7)	Filter Data(7)	Syn_OUT_EN(3)	Filter Data(7)
175	T6	I/O	Filter Data(8)	Filter Data(8)	Filter Data(8)	Filter Data(8)	Filter Data(8)	Filter Data(8)	Filter Data(8)	Filter Data(8)	DSP_OUT(0)	Filter Data(8)
176	T7	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
177	T8	I/O	Filter Data(9)	Filter Data(9)	Filter Data(9)	Filter Data(9)	Filter Data(9)	Filter Data(9)	Filter Data(9)	Filter Data(9)	DSP_OUT(1)	Filter Data(9)
178	T9	I/O(D3)	Filter Data(10)	Filter Data(10)	Filter Data(10)	Filter Data(10)	Filter Data(10)	Filter Data(10)	Filter Data(10)	Filter Data(10)	DSP_OUT(2)	Filter Data(10)
179	T10	I/O	Filter Data(11)	Filter Data(11)	Filter Data(11)	Filter Data(11)	Filter Data(11)	Filter Data(11)	Filter Data(11)	Filter Data(11)	DSP_OUT(3)	Filter Data(11)
180	T11	I/O	COMP_SELECT1	COMP_SELECT2	COMP_SELECT3	COMP_SELECT4	COMP_SELECT5	COMP_SELECT6	COMP_SELECT7	COMP_SELECT8	DSP_OUT(4)	BIRQX*
181	T12	GND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND
182	T13	I/O	AUX1	AUX1	AUX1	AUX1	AUX1	AUX1	AUX1	AUX1	DSP_OUT(5)	CCLK OUT
183	T14	I/O	AUX2	AUX2	AUX2	AUX2	AUX2	AUX2	AUX2	AUX2	DSP_OUT(6)	AD1
184	T15	I/O(D7)	AUX3	AUX3	AUX3	AUX3	AUX3	AUX3	AUX3	AUX3	DSP_OUT(7)	AD17
185	T16	SGCK3(I/O)	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1	CONTROL1
186	T17	I/O	AUX5	AUX5	AUX5	AUX5	AUX5	AUX5	AUX5	AUX5	DSP_OUT(9)	DONE_IN
187	T18	I/O	AUX6	AUX6	AUX6	AUX6	AUX6	AUX6	AUX6	AUX6	DSP_OUT(10)	REPROGRAM*
188	U1	PGCK4(I/O,A1)	AUX7	AUX7	AUX7	AUX7	AUX7	AUX7	AUX7	AUX7	DSP_OUT(11)	BSYSCLK
189	U2	TDO	U2-A2	U3-A2	U4-A2	U5-A2	U6-A2	U7-A2	U8-A2	U9-A2	U10-A2	JM8
190	U3	I/O(D0,DIN)	U2-T4	U3-T4	U4-T4	U5-T4	U6-T4	U7-T4	U8-T4	U9-T4	THRESH	PROM-DATA
191	U4	I/O	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	Syn_OUT_EN(0)	DSP_OUT(12)	CS1*
192	U5	I/O	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	Syn_OUT_EN(1)	DSP_OUT(13)	CS2*
193	U6	I/O	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	Syn_OUT_EN(2)	DSP_OUT(14)	CS3*
194	U7	I/O	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	Syn_OUT_EN(3)	DSP_OUT(15)	CS4*
195	U8	I/O	AUX8	AUX8	AUX8	AUX8	AUX8	AUX8	AUX8	AUX8	DSP_OUT(16)	CS5*
196	U9	I/O(RS*)	AUX9	AUX9	AUX9	AUX9	AUX9	AUX9	AUX9	AUX9	START_CLK(J5)	CS6*
197	U10	I/O(D4)	AUX10	AUX10	AUX10	AUX10	AUX10	AUX10	AUX10	AUX10	MCLK_OUT(J5)	CS7*
198	U11	I/O	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8	LED9	CS8*
199	U12	I/O(D5)	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	WRITE_ENABLE	DSP_OUT(17)	WRITE_ENABLE
200	U13	I/O	SPEC_SEL	SPEC_SEL	SPEC_SEL	SPEC_SEL	SPEC_SEL	SPEC_SEL	SPEC_SEL	SPEC_SEL	DSP_OUT(18)	SPEC_SEL
201	U14	I/O	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	OUT_EN(2)	DSP_OUT(19)	OUT_EN(2)
202	U15	I/O	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	OUT_EN(3)	DSP_OUT(20)	OUT_EN(3)
203	U16	PGCK3(I/O)	START	START	START	START	START	START	START	START	START	START
204	U17	DONE	DONE_IN	DONE_IN	DONE_IN	DONE_IN	DONE_IN	DONE_IN	DONE_IN	DONE_IN	DONE_IN	PROM-CE
205	U18	I/O	ACC_EN	ACC_EN	ACC_EN	ACC_EN	ACC_EN	ACC_EN	ACC_EN	ACC_EN	DSP_OUT(21)	ACC_EN
206	V1	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	CCLK	PROM-CLK
207	V2	I/O(RCLK-BUSY*/RDY)	AUX11	AUX11	AUX11	AUX11	AUX11	AUX11	AUX11	AUX11	COMP_SELECT8	BSYSRESET*
208	V3	I/O(D1)	RT1	RT2	RT3	RT4	RT5	RT6	RT7	RT8	RT9	CS9*
209	V4	I/O	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	OUT_TRIG(0)	DSP_OUT(22)	OUT_TRIG(0)
210	V5	I/O	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	OUT_TRIG(1)	DSP_OUT(23)	OUT_TRIG(1)
211	V6	I/O	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	OUT_TRIG(2)	COMP_SELECT1	OUT_TRIG(2)
212	V7	I/O(D2)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	OUT_TRIG(3)	COMP_SELECT2	OUT_TRIG(3)
213	V8	I/O	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	OUT_TRIG(4)	COMP_SELECT3	OUT_TRIG(4)
214	V9	I/O	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	OUT_TRIG(5)	COMP_SELECT4	OUT_TRIG(5)
215	V10	I/O	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	OUT_TRIG(6)	COMP_SELECT5	OUT_TRIG(6)
216	V11	I/O	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN	FIR_EN
217	V12	I/O(CSO*)	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN	FFT1_EN
218	V13	I/O	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN	FFT2_EN
219	V14	I/O	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN	SPEC_EN
220	V15	I/O	AUX12	AUX12	AUX12	AUX12	AUX12	AUX12	AUX12	AUX12	COMP_SELECT6	RT10
221	V16	I/O	AUX13	AUX13	AUX13	AUX13	AUX13	AUX13	AUX13	AUX13	COMP_SELECT7	RD10
222	V17	I/O(D6)	RD1	RD2	RD3	RD4	RD5	RD6	RD7	RD8	RD9	DBEN
223	V18	PROG*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	REPROGRAM*	PROM-OE/RESET*

Designator_ch	c	Signal Name	STATE
	*		FUNCTION
	*		MCLK
	*		MCLK2
	*		MRST
	*		start*
	*		start2*
	*		istate
	*		count
	*		count2
	*		fir_en
	*		fir_en2
	*		fir_count
	*		fir_count2
	*		fft1_en
	*		fft1_en2
	*		fft1_count
	*		fft1_count2
	*		fft2_en
	*		fft2_en2
	*		fft2_count
	*		fft2_count2
	*		spec_en
	*		spec_en2
	*		spec_count
	*		spec_count2
	*	*MULTIBEAM*	
	*		Multiplier result
	*		Register OUT
	*		ACC OUT CH.A
	*		ACC OUT CH.B
U1-ch0,ch2	*		INPUT MUX INPUT : A
U1-ch0,ch2	*		INPUT MUX FIR : B
U1-ch0,ch2	*		INPUT MUX FFT : C
U1-ch0,ch2	*		INPUT MUX SPEC : D
U1-U8		INPUT_DATA_S	INPUT DATA SEL
U1-ch0,ch2	*		DATA MUX LOC. : A
U1-ch0,ch2	*		DATA MUX FIR : B
U1-ch0,ch2	*		DATA MUX FFT : C
U1-ch0,ch2	*		DATA MUX SPC : D
U1-U8		INPUT_DATA_S	INPUT DATA SEL
U1-ch0,ch2	*		FFT INPUT 1st Ibus
U1-ch0,ch2	*		FFT INPUT 1st Qbus
U1-ch0,ch2	*		FFT INPUT 2nd Ibus
U1-ch0,ch2	*		FFT INPUT 2nd Qbus
U1		FFT_SEL	FFT MUX
			FIR IQ REG MUX: MEM
			FIR IQ REG MUX: REG
		IQ_RAM_SEL	IQ RAM-REG MUX
U10		OUT_TRIG(7)	IO REG TRIG
U10	*	OUT_EN(0)	write_en
U10	*	OUT_EN(1)	spec_sel
U10	*	OUT_EN(2)	Accumulation EN*
	*		ACCUM Reg Latch
U1		FFT_SEL	FFT MUX
U2		FFT_SEL	FFT MUX
U3		FFT_SEL	FFT MUX
U4		FFT_SEL	FFT MUX
U5		FFT_SEL	FFT MUX
U6		FFT_SEL	FFT MUX
U7		FFT_SEL	FFT MUX
U8		FFT_SEL	FFT MUX
U1~U4		OUT_EN(0)	FIR I,Qch OUT EN*
U5~U8		OUT_EN(0)	FIR I,Qch OUT EN*
U1,U5		OUT_EN(1)	FFT 1st Ich OUT EN*
U2,U6		OUT_EN(1)	FFT 1st Ich OUT EN*
U3,U7		OUT_EN(1)	FFT 1st Ich OUT EN*
U4,U8		OUT_EN(1)	FFT 1st Ich OUT EN*
U1~U4		FIR_SEL	fir_I_sel
U1~U4		FIR_SEL	fir_q_sel
U5~U8		FIR_SEL	fir_I_sel
U5~U8		FIR_SEL	fir_q_sel
U10	*	OUT_TRIG(0)	FIR Ich OUT TRIG
U10	*	OUT_TRIG(1)	FIR Qch OUT TRIG
U10	*	OUT_TRIG(2)	FFT 1st Ich OUT TRIG
U10	*	OUT_TRIG(3)	FFT 1st Qch OUT TRIG
U10	*	OUT_TRIG(4)	Synth Ich OUT TRIG
U10	*	OUT_TRIG(5)	Synth Qch OUT TRIG
U10	*	OUT_TRIG(6)	Synth SPECTRUM OUT TRIG
		=start	INPUT Register TRIG
			ram_count_en
			add_store_en=OUT_TRIG(3)
			iq_ram_load = start
			iq_ram_add(4)(iq_state)
			rd_current_add(3:0)
U1-ch0,ch2	*	RD_RAM_ADD	FIR Memory ADDRESS(1st lap)
			FIR Data
		RD_RAM_ADD	FIR Memory ADDRESS(2nd lap)
U1-ch0,ch2	*	MCLK	FIR OUT Register TRIG
			FIR Memory OUT
			FIR OUT Register OUT
U1-ch0,ch2	*	FFT_WN_ADD	FFT Wn Memory ADDRESS
			FFT Wn Memory OUT
U1-ch0,ch2	*		FFT Wn Register TRIG
			FFT Wn Register OUT
U1-ch0,ch2	*		FFT OUT Register TRIG
U1-ch0,ch2	*		Synth OUT Register TRIG
U1-ch0,ch2	*		FIR An OUT Register TRIG

Address is the same for both channel, but memory shuld be different.


Time
Designator.ch
c
Signal Name
STATE
FUNCTION
MCLK
MCLK2
MRST
start1*
start2*
istate
count
count2
fir_en
fir_en2
fir_count
fir_count2
fft1_en
fft1_en2
fft1_count
fft1_count2
fft2_en
fft2_en2
fft2_count
fft2_count2
spec_en
spec_en2
spec_count
spec_count2
BEAM SELECTOR
COMP OUT(MCLK2)
U9 COMP IN(MCLK)
COMP RESULT
MAX_CH(MCLK2)
SYN_EN(MCLK)U1~U8
U1~U8 SYN_OUT (M2)
MAX_ADD(MCLK)
INT REG(MCLK)
DISABLE
MAX_AMP
THRESH COMP
CH_REG
COMP_SEL OUT
DISABLE OUT
U10 Module IN
U10 I-REG TRIG
U10 I-REG SEL
DATA_TRIG
AI ROW REG
AI INFF
AI OUTFF
AI RAM OUT
PHASE SHIFTER
MC IN
MP IN
REG_OUT
ACC_OUT
SYN DATA REG TRIG
SYN DATA RAM WE
SYN DATA IQ SEL
MAX CH RAM ADD(1:0)
MC RAM WE
PHASE ROM IQ SEL
ADD_SUB
ACC_EN
OUT_REG TRIG EN
ROW_RAM_ADD(1:0)
ROW_RAM_I_WE
ROW_RAM_Q_WE

APPENDIX 設計データソフトの使用方法

1. 開発環境

付録の設計データのソフトウェアをDATテープに収録した。本設計データを使用するためには以下の開発用CADソフトが必要となる。

1. Mentor Graphics 社 Design Architect (A1-Fバージョン以上)
2. Mentor Graphics 社 System1076-Complier、Autologic

(あるいは、VHDLを論理合成できるツール。 ただし、本VHDLソースコードは1部Mentor用に記述されているので、他のツールを使用する場合、記述の変更が必要。例: library mgc_portable)

3. Xilinx 社 XACT5.0以上のライブラリ MentorGraphics オプション付き。

また、ATRでの開発環境は以下のとおりである。

1. エンジニアリングワークステーション HP9000/700 Model 735/125, RAM208MB, OS HP-UX9.05
2. Mentor Graphics A1-F
   Design Architect
   System1076-Compiler
   Autologic
   Autologic Blocks
3. Xilinx XACT5.1
4. ターゲットデバイス (XC4025PG223-5)

これから得られたネットリストをFPGAに配置配線するために、以下のCADソフトウェアを用いた。

1. 旧Neocad社 (現Xilinx社) FPGA Foundary Version.7.0

2. DATテープからのレストア

tarコマンドを使用してレストアする。
スーパユーザでダウンロードするディレクトリに移動してからtarコマンドで行なう。

```
# cd xxx
# tar xvf /dev/rmt/0mn
```

ダウンロード後にはcomponents5というディレクトリが作成されており、そのディレクトリを

/users3/design5/mentor/

の下にリンクあるいは移動する。つまり回路図から

"/users3/design5/mentor/component5"

が参照できなくてはならない。各データをダウンロードするために必要なディスク容量は
DBFマルチビーム: 約40MBytes
BSCMAアダプティブ部: 約11MBytes
Self-Beam-Steering部: 約13MBytes
である。

3. 設計データの展開

設計データは本体の回路図とその構成部品の2つに分かれている。
部品 (component5のディレクトリの下の回路ディレクトリ) は以下のディレクトリに置かなければならない。

"/users3/design5/mentor/component5"

また、Xilinx社のXACT5.0バージョンのxc4000シリーズ用ライブラリを

$LCA/xc4000 ($LCAは任意のディレクトリ)

の下に存在させなければならない。