Chapter 3

Fast Prototyping System

In this chapter, we will introduce the development environment. The environment as shown in Figure 3.1 includes a fast prototyping platform (Aptix® System Explorer) with several specific modules, a high speed work station, a digital to analog converter (DAC), an analog to digital converter (ADC), a logic analyzer (LA), an oscilloscope, and some PCs.

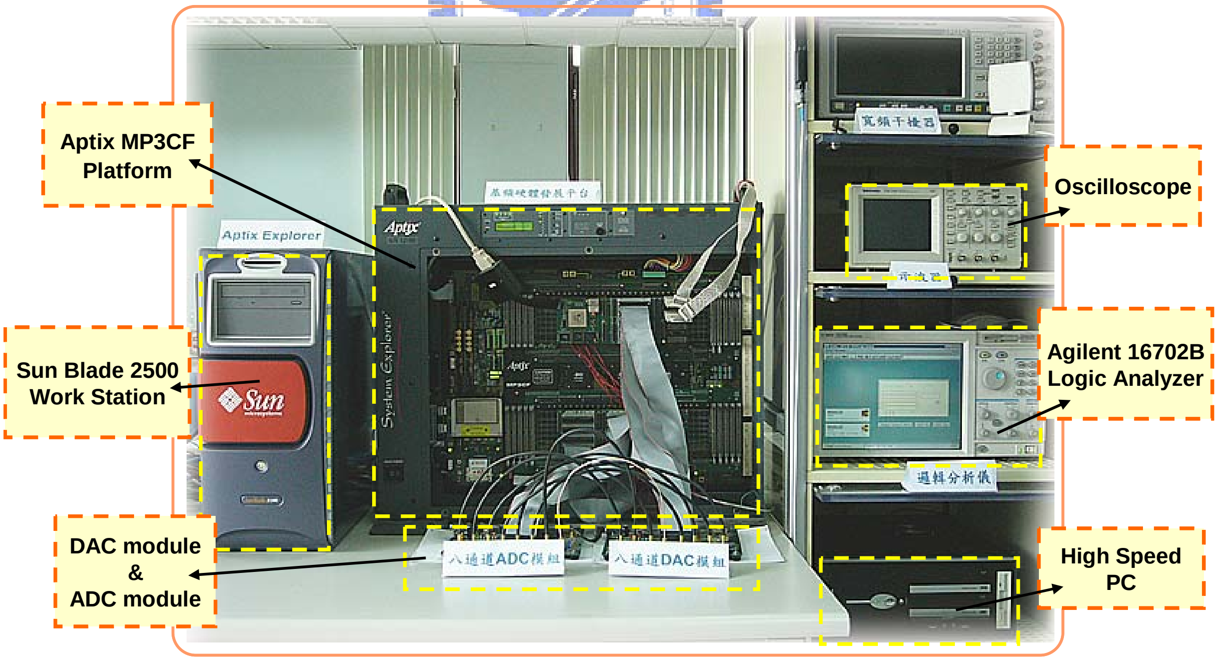


Figure 3.1: Development environment of fast prototyping system

We first give an overview of the Aptix® platform, which includes software and hardware. Moreover, we will look into those modules installed on the platform as shown in Figure 3.2. There are mainly FPGA modules and a DSP evaluation module (EVM) which share the major task of implementation. In particular, we will show how they communicate with each other on the platform. In addition to FPGA and DSP, USB module is also an important part for providing USB 2.0 interface between the PC and Aptix® platform. The main devices beside the platform are ADC and DAC, which can make the development system more close to a real communication system. Finally, we will introduce the debugging tools.

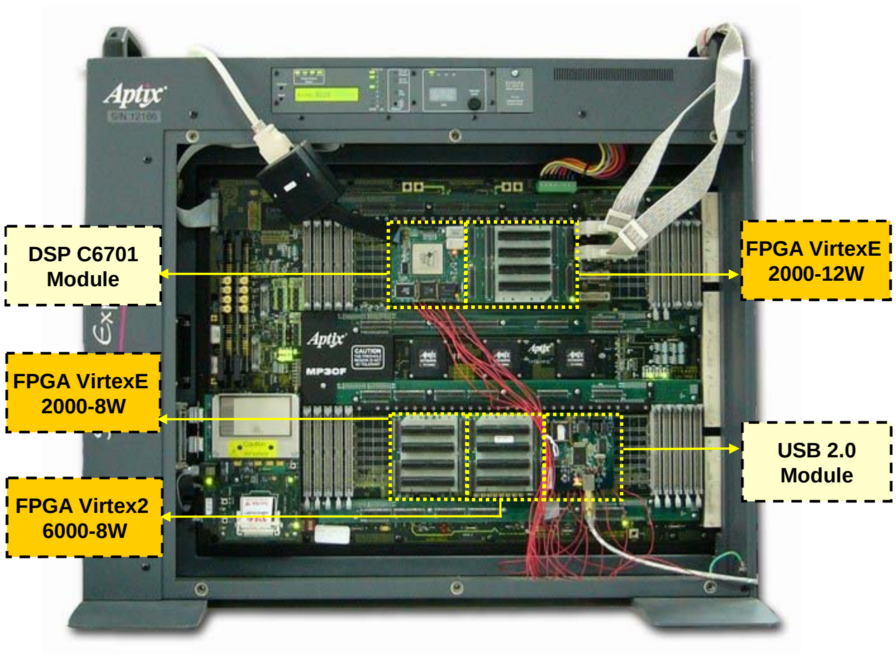


Figure 3.2: Modules installed on Aptix MP3CF platform

3.1 Aptix® System Explorer

Under the trend of System on Chip (SoC) and the concept of time-to-market, Aptix® Corporation has developed a series of fast prototyping system named MPx, which provides a total solution of real-time verification and integration for industry and high-performance functional simulation for application specific integrated circuit (ASIC) designer so as to achieve the goal of time-to-market. In our laboratory, we choose Aptix® System Explorer MP3CF as our fast prototyping system. Combining MP3CF with an LA through Ethernet, we can build up a complete hardware fast prototyping system, where we can easily develop the adaptive 4x4 MIMO-OFDM system and conveniently verify our design in FPGA and DSP.

The Aptix® MP3CF System Explorer™ contains two parts, hardware platform called MP3CF FPCB and software called Explorer, on which we will give more introduction in later subsections.

3.1.1 Hardware: MP3CF Platform

Aptix® MP3CF Platform consists of several functional units, such as the onboard micro-controller, the clock generator, some re-programmable inter-connect chips called field programmable interconnect components (FPIC), the main motherboard called field programmable circuit board (FPCB), and some flexible input/output (I/O) buses [26] as illustrated in Figure 3.3.

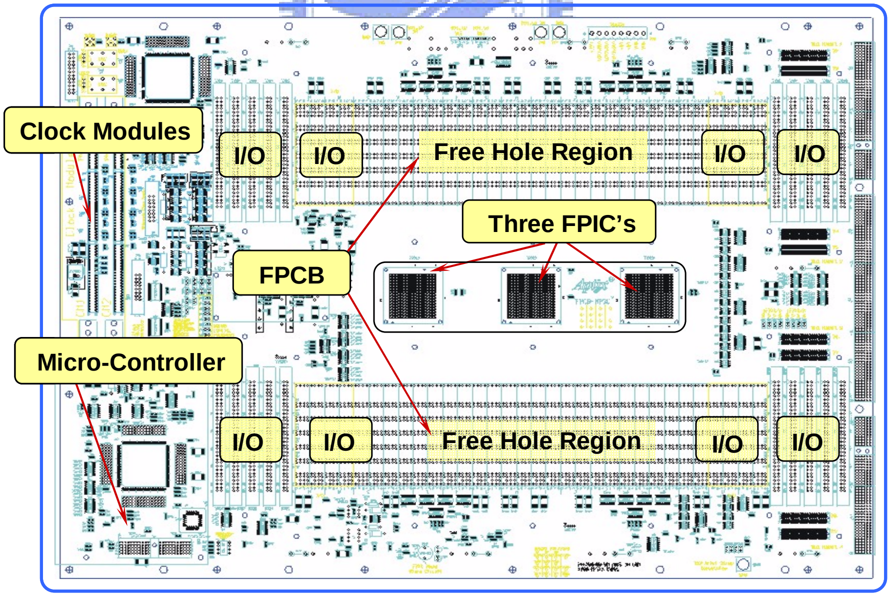


Figure 3.3: Aptix® MP3CF platform

Micro-controller mainly takes charge of the operation of the whole platform, such as the control of booting sequence and storing or loading the design of circuit through flash memory; Clock generator provides system clock, and supports eight different clock sources from outside; FPIC is responsible for the inter-connect of all modules; FPCB is the place where modules can be installed; I/O bus is the bridge between Aptix® platform and devices outside.

FPCB and FPIC are two major core techniques of Aptix® MP3CF platform. Actually, FPCB is a large programmable circuit board containing 3520 freeholes. Each freehole can accept a pin and connect to the FPIC chip which is a programmable routing chip. Each freehole can be routed by FPIC when several modules are working. There are three FPIC chips located on the center of Aptix® MP3CF, and each contains 1024 I/O pins and defines the way to route among all freeholes so that each module can communicate bi-directionally.

Freeholes can be divided into three categories. The first one is special pins, which supply modules with the power and clock. The second one is I/O pins, which provide the connection between modules onboard or devices outside. The last one is swappable pins, which satisfy the specification of transistor-transistor logic (TTL) voltage level and become the interface between modules to connect with each other.

Aptix® MP3CF is powerful and capable of easy expansion and high integration. It not only supports modules produced by Xilinx Corporation and Altera Corporation, but also those fitting the definition of freehole pins. By the right definition, we can install modules developed by other companies on Aptix® MP3CF through an adapter. For example, we developed a DSP C6701 EVM by using the core chip TMS320C6701 DSP of Taxas Instruments (TI) and also a CYPRESS USB 2.0 module by using the core chip of CYPRESS CY7C68013, both of them being not the products from Xilinx or Altera. Therefore, by the usage of the adapter, we can integrate different modules on Aptix® MP3CF and make the system more flexible and powerful.

3.1.2 Software: Explorer

The software (called *Explorer*) provides an easy-to-use, consistent user interface which displays commands through a series of pull-down menus. The main design flow is described as follows and illustrated in Figure 3.4.

(1) **Import Design into Explorer**

Explorer needs to be informed about the netlist files that we are using in the design including Top-level netlist, Component netlist, and Pinmap file. Top-level netlist is an electronic design interchange format (EDIF) file containing connectivity information between the different components that will be mounted on the MP3CF FPCB. Component netlists are EDIF files containing major design information in each component. All EDIF files can be generated by electronic design automation (EDA) tools that can support synthesis, such as Xilinx Foundation we adopt. Finally, we have to identify the Pinmap file used in the design to assign packages, pins, and other information to those parts.

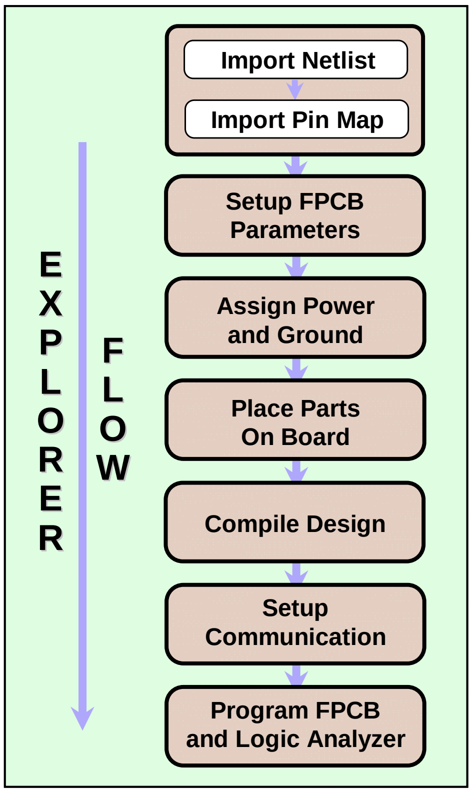


Figure 3.4: Explorer flow

(2) **Setup FPCB Parameters**

Explorer can support several different FPCBs. We need to specify which FPCB we are using to develop.

(3) **Assign Power and Ground**

Some physical parameters of the design need to be set up, such as power and ground nets.

(4) **Place Parts on Board**

We need to place our design components in their correct positions on the coordinate system. There will be a Board view window helping us move a component onto the right place of FPCB by dragging the component to the desired place with a mouse.

(5) **Compile Design**

The compilation process first maps the FPCB and then maps the existing I/O, clock, bus and FPGA nets to MP3CF hardware. Using the result of FPCB mapping, compilation continues with FPGA place-and-route which will run for all FPGAs in the design. Once the FPGA place and route has been completed successfully, compilation conducts the FPCB routing. The FPCB router routes the FPICs with all nets in the design mapped to the FPGAs. In general, place-and-route is the most time-consuming process of all.

(6) **Setup Communication**

In this process, we need to do some configurations about communication to program the board and devices. For hardware (FPCB board), we need to specify communication method, address for the method, and whether the flash is to be programmed or not when downloading. For debug (LA), we need to identify communication method, address for the method, and which probing pod of the LA is to be connected with.

(7) **Program FPCB and LA**

Finally, we can download our design onto FPCB and probing information to the LA, and start to verify our system design.

3.2 FPGA Module

In our fast-prototyping system, we use several FPGA modules mounted on Aptix® MP3CF platform to implement our communication system. In the following sub-sections, we will give an overview of our FPGA modules. Then we will show the design flow of FPGA.

3.2.1 FPGA Overview

The demand for more complex programmable hardware is constantly growing to meet the formidable industry requirement. The major categories of programmable hardware are programmable logic device (PLD) and FPGA. A PLD consists of micro-cells and a central inter-connection logic. Typical PLD applications are "glue logic" for connecting other ASICs. On the other hand, FPGAs consist of even more complex logic block on one chip. Typical applications are central control units (CPU) and DSPs up to very complex SoC design. Therefore, we adopt some FPGA modules to realize our communication system. Generally, FPGA can be categorized into three types by its structure:

1. **Look-up-table (LUT)**: Xilinx, Altera, AT&T
2. **Multiplexer**: Actel, Quicklogic
3. **Transistor array**: Cross point

If we focus on its programming architecture, there are two major types:

1. **SRAM**: Xilinx, Altera, AT&T, Atmel
2. **Anti-fuse**: Actel, Cypress, Quicklogic

Static random access memory (SRAM) type has a merit of being able to program repeatedly while Anti-fuse type has the feature of one time programmable (OTP). Anti-fuse type can offer security for design but cannot be modified further.

Compared to ASIC, FPGA has lower performance apparently, especially on power consumption and maximum supportable speed. However, as the technique of semiconductor industry grows, FPGA becomes more and more competitive to ASIC. Actually, FPGA has more integration ability and flexibility than ASIC, and undoubtedly, is the best candidate component for a fast-prototyping system.

3.2.2 FPGA Design Flow

In our design, we choose Xilinx Foundation software as the development tool for the first half of the design flow. The second half is done on a workstation with Explorer. Figure 3.5 is the main FPGA design flow and later we will give more information about the flow.

(1) **Design Entry**

In general, EDA tools are needed to develop register transfer level (RTL) codes by appropriate methodologies. In Xilinx Foundation, it supports three methods: HDL (hardware description language) Editor, Schematic Flow, and FSM (finite state machine) Editor. HDL Editor allows us to edit source files directly like VHDL (very high speed integrated circuit hardware description language) [27]-[30] and Verilog [31], which are the most common HDLs in use today. Schematic Flow is another choice to create our source files by drawing the scheme with underlying HDL macros. FSM Editor allows us to edit by timing state diagram, which is suitable for realization controller, such as memory access controller.

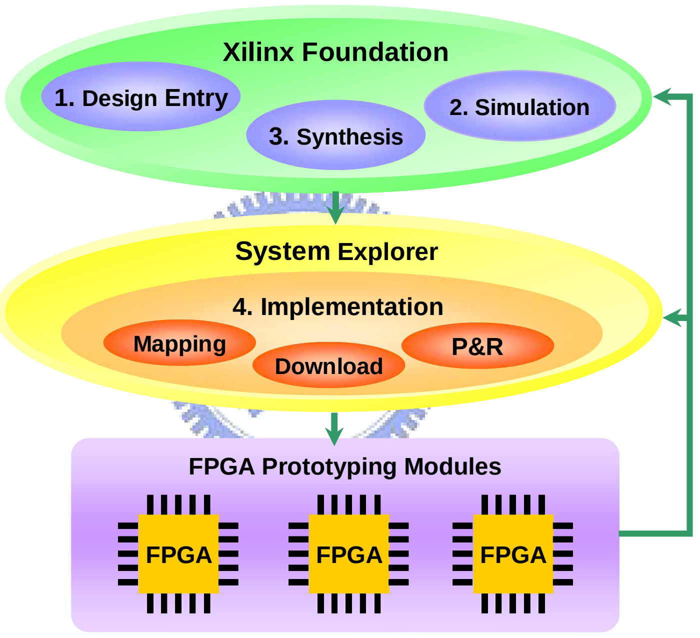


Figure 3.5: FPGA design flow

(2) **Synthesis**

After completing editing RTL source files, we need to translate them into gate level called netlist files, which only contains information of logic gates and inter-connections. Although, there are many EDA tools proficient in synthesis, such as Synopsys and Sinplicity, we choose to use Xilinx Foundation for synthesis for the sake of convenience.

(3) **Simulation**

Design verification is an important aspect of each project design. Before implementing our circuit in the target device, it is a good idea to simulate and verify the circuit. The most common verifications are functional simulation and timing simulation.

A. Functional Simulation

Functional simulation can be done after the schematic has been entered or a HDL file has been created and synthesized. Functional simulation gives information about the logic operation of the circuit, but it does not provide any information about timing delays.

B. Timing Simulation

The timing simulation will give us detailed information about the time it takes for a signal to pass from one gate to the other (gate delay) and gives information on the circuit's worst-case conditions. The total delay of a complete circuit will depend on the number of gates the signal sees and on the way the gates have been placed in the FPGA.

(4) **Implementation**

The implementation is typically done after the design has been verified by functional simulation. The implementation tools will translate the netlist (schematic, HDL), place and route the design in the target device and generate a bitstream that can be downloaded into the device.

(5) **Download to Aptix® Explorer MP3CF**

After the process of implementation, we can download our design into hardware platform. To verify that signals are really working properly in circuit, we can use the LA to debug. Once the result does not match what we expect, we need to come back to modify our design and go through the whole design flow again. That is to say, iterative tests are required until we obtain the results we want.

3.3 'C6701 DSP EVM

Digital signal processors, such as TMS320 family of processors, are used in a wide range of applications, from communications and controls to image and speech processing. They are found in cellular phones, fax/modems, disk drivers, radio, and so on. Texas Instrument recently introduced the TM320C6x processor, based on the very-long-instruction-word (VLIW) architecture. This newer architecture supports features that facilitate the development of efficient high-level language compilers. The TMS320C67x DSPs are the floating-point DSP family in the TMS320C6000E DSP platform. We choose TMS320C6701 as our core chip on DSP EVM to implement our adaptive 4x4 MIMO OFDM system. Later, we will give an overview of the core chip on DSP EVM. Then we will introduce the architecture of EVM. Finally, we will show the design flow about DSP.

3.3.1 TMS320C6701 DSP Overview

In the following sub-section, we will introduce the architecture of the core chip, TMS320C6701 DSP, which can be divided into three parts, including a CPU, memories, and peripheral components [32][33].

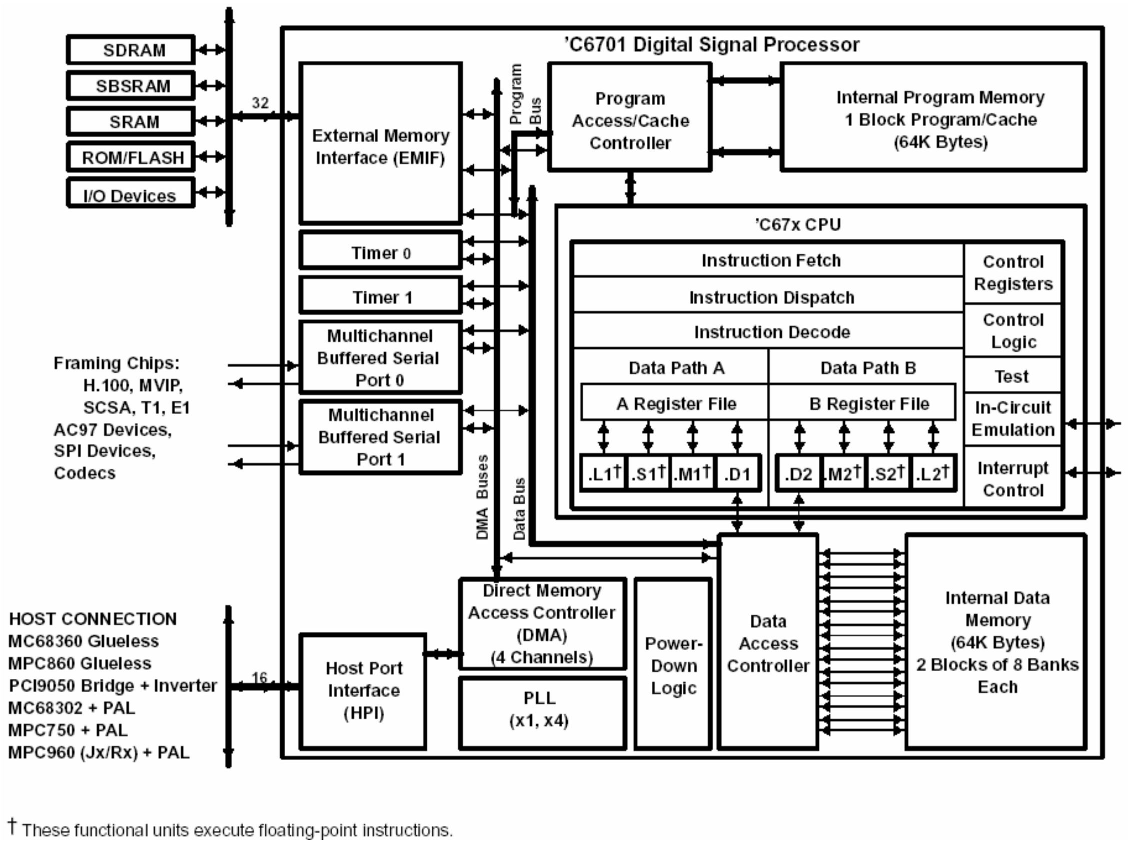
3.3.1.1 'C6701 DSP CPU

TMS3206701 DSP consists of eight independent functional units divided into two data paths A and B, as shown in Figure 3.6. Each path has the following units:

1. **.M unit**: dedicated for multiply operations; providing two 16-bit variables to multiply and the output is 32-bit.
2. **.L unit**: performing a general set of arithmetic, logical functions, such as AND, OR, and NOT.
3. **.S unit**: performing branch and bit manipulation functions.
4. **.D unit**: responsible for all data transfer between the register files and the memory, and providing either linear- or circular-addressing.

Each functional unit can read directly from or write directly to the register file within its own path. Each path includes a set of sixteen 32-bit registers, A0 through A15 and B0 through B15. Units ending in 1 write to register file A, and units ending in 2 write to register file B.

The ‘C67x CPU executes all TMS320C62x™ DSP fixed-point instructions. In addition to the ‘C62x DSP fixed-point instructions, the six out of eight functional units (.L1, .M1, .D1, .D2, .M2, and .L2) also execute floating-point instructions.



'C6701 Digital Signal Processor
SDRAM
SBSRAM
SRAM
ROM/FLASH
I/O Devices
32
External Memory Interface (EMIF)
Program Bus
Program Access/Cache Controller
Internal Program Memory 1 Block Program/Cache (64K Bytes)
'C67x CPU
Instruction Fetch
Instruction Dispatch
Instruction Decode
Control Registers
Control Logic
Test
In-Circuit Emulation
Interrupt Control
Data Path A
Data Path B
A Register File
B Register File
.L1† .S1† .M1† .D1
.D2 .M2† .S2† .L2†
Timer 0
Timer 1
Multichannel Buffered Serial Port 0
Multichannel Buffered Serial Port 1
Framing Chips: H.100, MVIP, SCSA, T1, E1
AC97 Devices,
SPI Devices,
Codecs
DMA Buses
Data Bus
HOST CONNECTION
MC68360 Glueless
MPC860 Glueless
PCI9050 Bridge + Inverter
MC68302 + PAL
MPC750 + PAL
MPC960 (Jx/Rx) + PAL
16
Host Port Interface (HPI)
Direct Memory Access Controller (DMA) (4 Channels)
PLL (x1, x4)
Power-Down Logic
Data Access Controller
Internal Data Memory (64K Bytes) 2 Blocks of 8 Banks Each
† These functional units execute floating-point instructions.

Figure 3.6: Architecture of TMS320C6701 DSP

3.3.1.2 ‘C6701 DSP Memory

‘C67x DSP uses 32-bit for addressing, which the memory can theoretically be accessed to a range of 4 Gbytes. The arrangement of memory can be shown in Figure 3.7, including 64 Kbytes internal program memory, 64 Kbytes internal data memory, and 52 Mbytes external memory, and still some remaining memory for the control of peripherals. Later, we will give more information about internal memories.

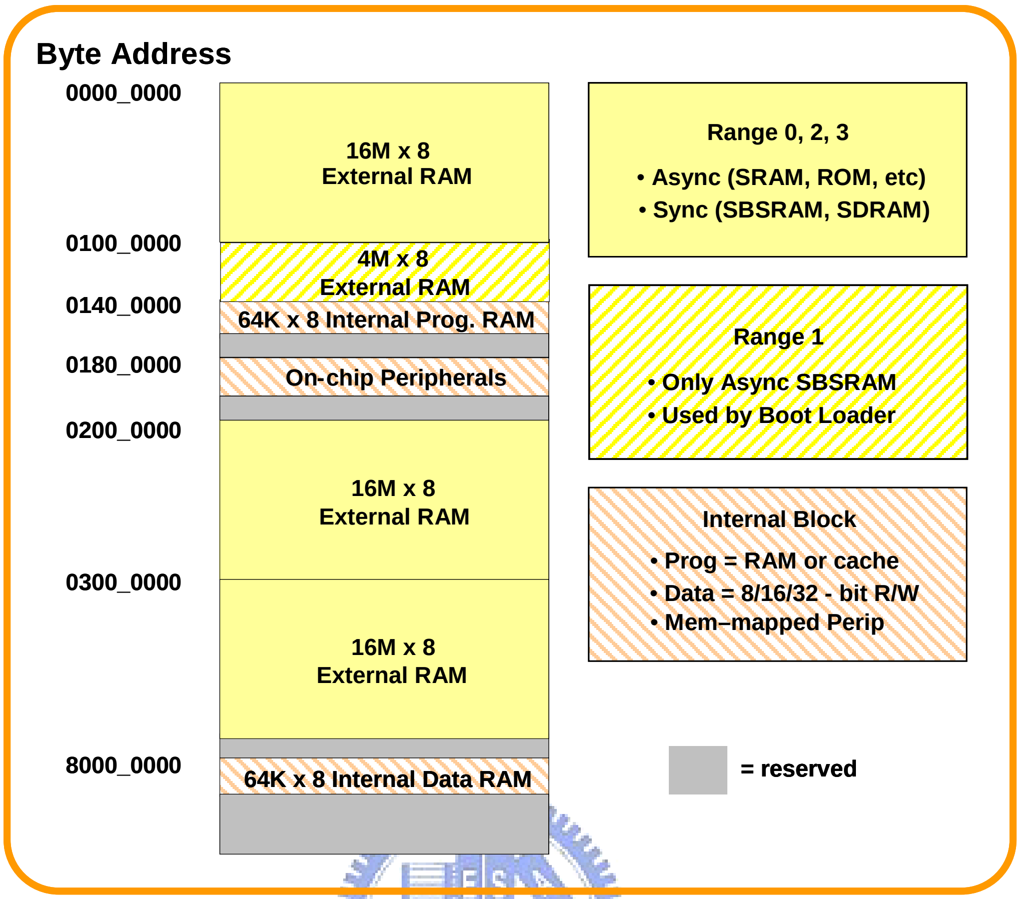


Figure 3.7: Memory mapping of TMS320C6701 DSP

(1) **Internal Program Memory**

The memory modes are decided by program memory controller (PMEMC), and the possible modes are as follows.

A. Cache Mode

In cache mode, all internal program memory is used as cache, and direct memory access (DMA) controller cannot access the memory.

B. Mapped Mode

In mapped mode, memory operation map mode can further divided into Map 0 and Map 1. When defined as Map 0, address from 0x01400000h to 0x140FFFFh are used for program memory; when defined as Map 1, address from 0x00000000h to 0x0000FFFFh are used for program memory. In mapped mode, both CPU and DMA controller can access any address of the memory. If the CPU and DMA attempt to access the same block of momory at the same time, then the DMA is stalled until the CPU completes its accesses to that block. After the CPU access is complete, the DMA is allowed to access the memory.

(2) **Internal Data Memory**

The internal data memory is controlled by data memory controller (DMEMC). The 64 Kbytes of internal data random access memory (RAM) are organized as two blocks of 32 Kbytes. Both blocks are organized as eight 2 K banks of 16-bit half-words. Both the CPU and DMA controller can still simultaneously access data that resides in different banks within the same block without performance penalty.

3.3.1.3 ‘C6701 DSP Peripherals

The main peripheral components of ‘C67x DSP include DMA controller, host port interface (HPI), interrupt selector, and external memory interface (EMIF), which are summarized as follows.

(1) **Direct Memory Access Controller:**

The DMA controller transfers data between address ranges in the memory map without intervention by the CPU. The DMA controller has four programmable channels for DMA operation. In addition, a fifth (auxiliary) channel allows the DMA controller to service requests from the HPI.

(2) **Host Port Interface:**

As shown in Figure 3.8, the HPI is a parallel port through which a host processor can directly access the CPU’s memory space. Connectivity to the CPU’s memory space is provided through the DMA/EDMA (Enhanced DMA) controller. Both the host and the CPU can access the HPI control register (HPIC). The host can access the HPI address (HPIA) register, the HPI data (HPID) register, and the HPIC by using the external data and interface control signals.

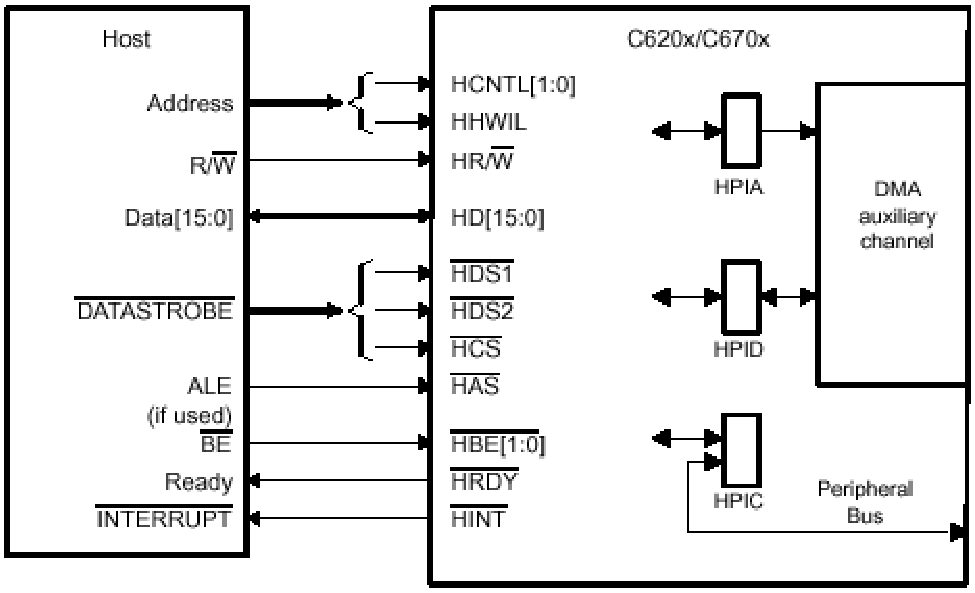


Figure 3.8: Host port interface of TMS320C6701 DSP

(3) **Interrupt Selector:**

The C6000 peripheral set has up to 32 interrupt sources. The CPU however has 12 interrupts available for use. The interrupt selector allows you to choose and prioritize which 12 of the 32 your system needs to use. The interrupt selector also allows you to effectively change the polarity of external interrupt inputs. There are three types of interrupts and they are differentiated by their priorities which are listed as follows [34].

A. RESET

The reset interrupt has the highest priority and corresponds to the RESET signal.

B. Non-maskable Interrupts (NMI)

The nonmaskable interrupt is the interrupt of second highest priority and corresponds to the NMI signal.

C. Maskable Interrupts

The lowest priority interrupts are interrupts 4–15. They correspond to the INT4–INT15 signals.

(4) **External Memory Interface:**

The external memory interfaces of the ‘C6701 support a glueless interface to a variety of external devices, including:

A. Pipelined synchronous-burst SRAM (SBSRAM)
B. Synchronous dynamic random access memory (SDRAM)
C. Asynchronous devices, including SRAM, read-only memory (ROM), and first in, first out (FIFO)
D. An external shared-memory device

The EMIF signals of the ‘C6701 are shown in Figure 3.9. The ‘C6701 provides separate clock and control signals for the SBSRAM and SDRAM interface. Asynchronous interface is supported on all CE spaces, but CE1 is used for asynchronous interface only.

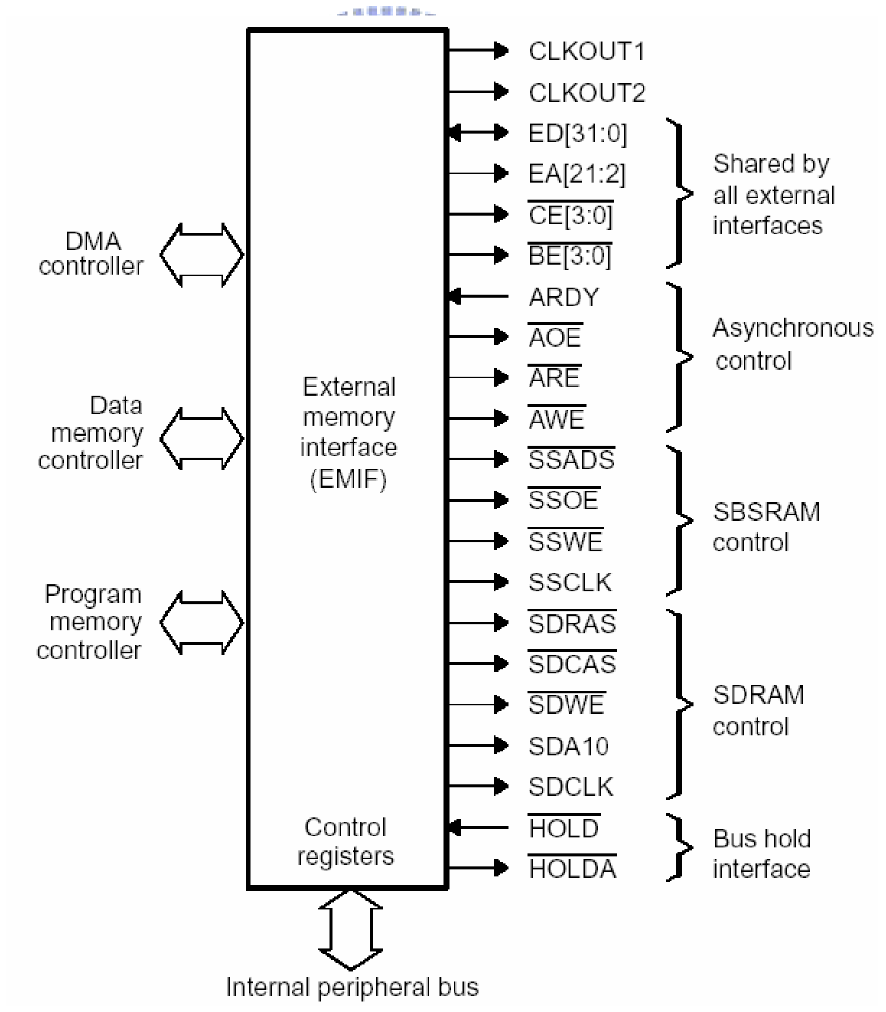


Figure 3.9: External memory interface of TMS320C6701 DSP

3.3.2 'C6701 DSP EVM Architecture

'C6701 DSP EVM shown in Figure 3.10 is developed to integrate with other modules on Aptix® platform so that we can come to the realization of an adaptive 4x4 MIMO-OFDM system. The EVM is applicable for Aptix® MPx series platform; it uses TMS320C6701 DSP as its core chip. The system clock is 132 MHz, and can be upgraded up to 167 MHz. Owing to having eight functional units in CPU, the DSP can perform 1056 mega floating-point operations per second (MFLOPS).

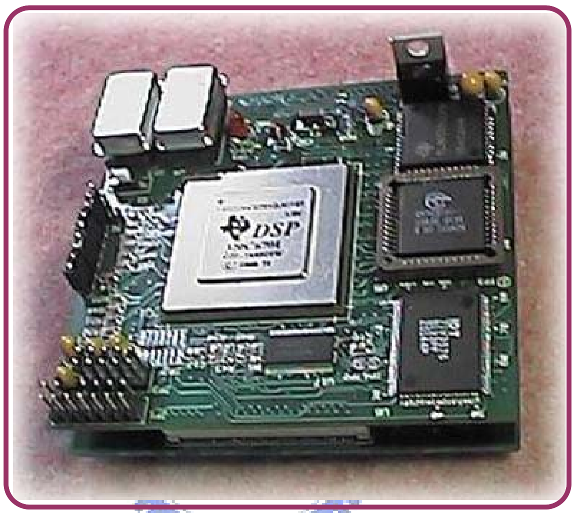


Figure 3.10: 'C6701 DSP EVM

The architecture of 'C6701 DSP EVM is shown in Figure 3.11, including TMS320C6701 DSP, flash memory, SBSRAM, universal asynchronous receiver/transmitter (UART), joint test action group (JTAG), and other interface circuits like transceiver and complex programmable logic device (CPLD). Later, we will give more information to what have not been mentioned.

(1) **Flash Memory:**

It is a nonvolatile read-only memory that is electronically erasable and programmable, and it has a capacity of 128 Kbytes. When completing our development, we can program the design into the flash memory. On the other hand, when we reset the DSP, it will automatically load the design from flash memory into internal program memory.

(2) **SBSRAM:**

SBSRAM works on the frequency of 132 MHz and has a capacity of 512 Kbytes. There are two working modes determine what it is used for, called Map 0 and Map 1. When Map 0 mode is set, it plays the role of program memory. When Map 1 mode is set, it is taken as general memory.

(3) **JTAG and UART:**

Both of them are interfaces of data transmission. JTAG is an interface compliant with IEEE 1149.1 standard interface, and it also connects with Innovate Integration Code Hammer PCI interface on PC to load the program from the software, Code Composer Studio (CCS). We can even stop the program and catch the values in memory through JTAG while debugging; UART is the other choice to connect with PC through RS-232 port.

(4) **Other Interface Circuit:**

CPLD offers four control signals to handle the connection with FPGA or other modules.

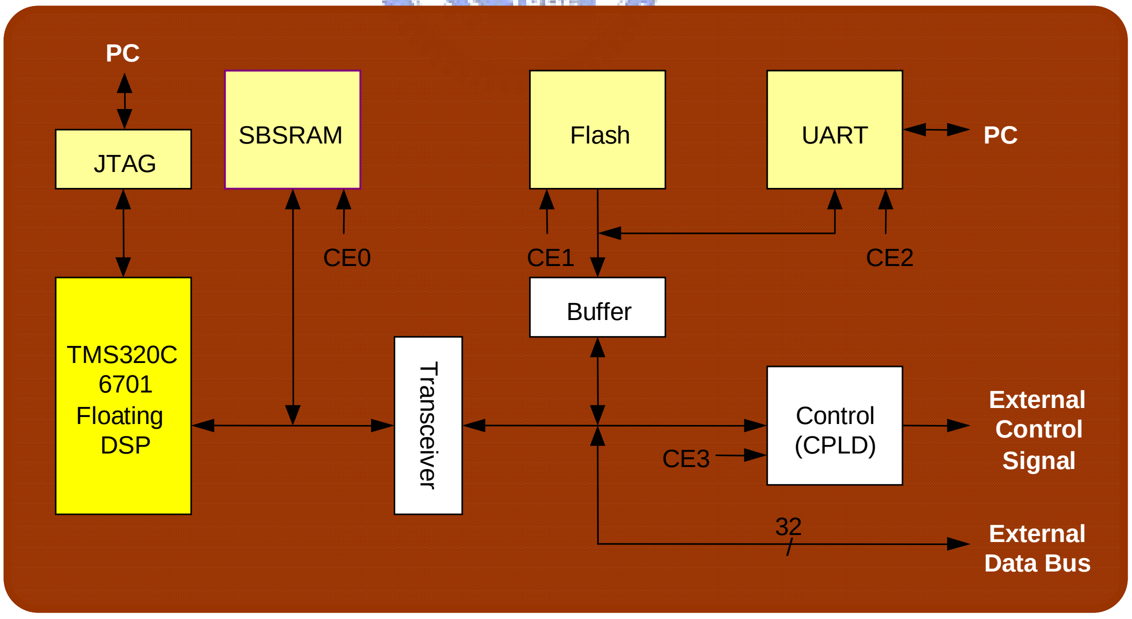


Figure 3.11: Architecture of ‘C6701 DSP EVM

3.3.3 DSP Design Flow

The Code Composer Studio (CCS) [35] provides an integrated development environment (IDE) to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. It also has graphical capabilities and supports real-time debugging, which enables us to develop our design efficiently. The DSP design flow with CCS can be separated into the following parts.

(1) **Create Project:**

First of all, we need to create a project, and add the necessary files for building the project. The most important files are source files, which can either be C source files (.c) or assembly source file (.asm). Then we also need Linker Command File (.cmd) and a run-time support library file (.lib). Last, we may need some header files (.h) to be included.

(2) **Code Generation and Options:**

Various options are associated with code generation tools, such as C compiler and linker. We can set up Compiler Option and Linker Option to do further configuration if we need, or we can just use the default setting in most cases.

(3) **Building and Running the project:**

After finishing code generation, we can build and run the project. In this process, it compiles and assembles all C files using *c16x* and assembles the assembly files using *asm6x*. The resulting object files are then linked with run-time library support file using *lnk6x*. This generates an executable file that can be loaded into 'C6701 processor and run. Then, we can load the program after a build.

(4) **Monitoring the Watch Window:**

Before monitoring the watch window, we need to verify that the processor is still running. After that, monitoring watch window allows us to change the value of a parameter or to monitor a variable we desire. Through monitoring, we can do debugging and regressive test until it works as we expect.

(5) **Correcting Program Errors:**

Once an error occurs, the error message will be listed and being a link directly to the line where the error occurs. After making the appropriate correction, we have to build, load, and run the program again to verify our results.

3.4 Communication Between FPGA and DSP

Having introduced FPGA and DSP EVM in the previous two sections, we will further give some ideas about the communication mechanism between them. Due to the discrepancy of working frequency, we need to define the timing specification of DSP while it connects with other modules, so that we can transmit or receive data without any errors. In DSP EVM, data transmission is not handled directly by DSP CPU, but taken charge by the four pieces of front processing board (from board 1 to board 4) as shown in Figure 3.12.

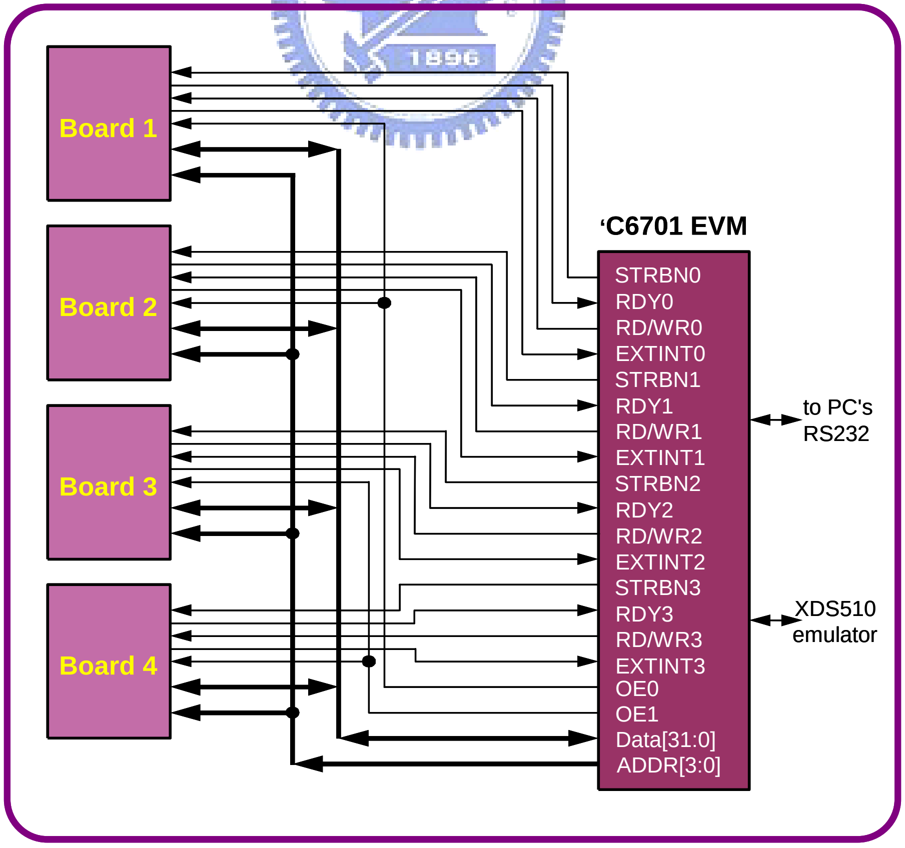


Figure 3.12: 'C6701 DSP EVM and its outer components

EXTINT0-EXTINT3 are interrupt signals, which is mapped to interrupt *INT4-INT7* of DSP CPU and can be requested to perform corresponding interrupts by other modules on Aptix® platform. In our design, whenever DSP is requested to work, we trigger *EXTINT0-EXTINT3* signals from FPGA module, which force DSP to perform the required task, and then transfer data back at the suitable time. Therefore, the goal of data transmission is achieved.

The timing diagram for DSP EVM to receive data from the outer module like FPGA is defined as in Figure 3.13. The duration of DSP clock is the reciprocal of the working frequency (duration tclk = 1/132 MHz). When *STRBN0/1/2/3* is on the falling edge and *RD/WR0/1/2/3* is on high level, it means that the front processing components are in the receiving state. Before the time t4 earlier than *STRBN0/1/2/3* coming back to high level, we must maintain the databus to be stable so that we can store data according the address at this moment.

On the contrary, if we need to transfer data to FPGA module from DSP EVM, the writing timing diagram can be followed in Figure 3.14. When both *STRBN0/1/2/3* and *RD/WR0/1/2/3* are on the falling edge, we can obtain the correct data at the time 1 tclk after *STRBN/0/1/2/3* is on the falling edge. The correct data can maintain a time-span t2 on the databus, and then the databus will return to a state of high-impedance. The followings are the parameters mentioned above or shown in Figures.

1. **tclk:** is the duration of a DSP clock. tclk = 1/132 MHz while working on the frequency of 132 MHz.
2. **t1:** is the time-span from the time that address line is ready to the falling edge of *STRBN0/1/2/3*.
3. **t2:** is the time-span for DSP to receive data.
4. **t3:** is the time-span from the falling edge of *STRBN0/1/2/3* to the time that address line is ready
5. **t4:** is the time that has to maintain the data on the databus.
6. **t5:** is the time-span that DSP can recognize Ready signal.
7. **10ns:** is the minimum propagation delay time caused by CPLD component.

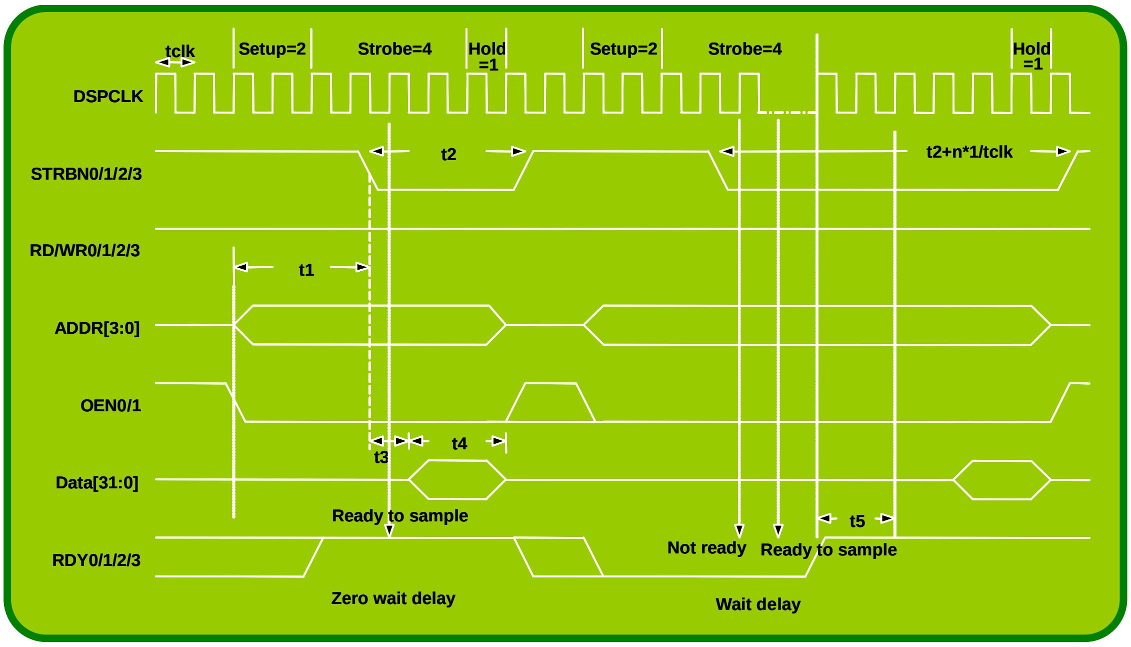


Figure 3.13: Read state timing diagram of ‘C6701 DSP EVM

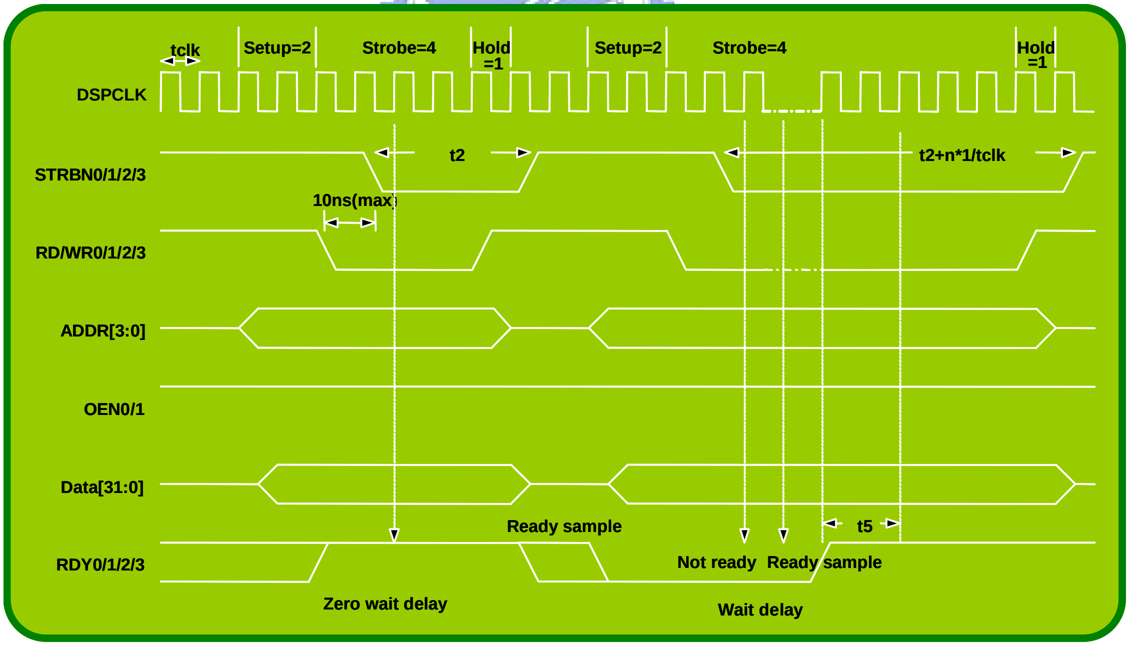


Figure 3.14: Write state timing diagram of ‘C6701 DSP EVM

3.5 USB 2.0 Module

USB 2.0 Module uses CYPRESS CY7C68013 [36] as its core chip as shown in Figure 3.15, which includes a 24 MHz 8051 and a 4 Kbytes FIFO. The maximum data rate can be up to 480 Mbps. The FIFO provides the interface between USB 2.0 module and C6701 EVM. Figure 3.16 shows a diagram of the USB 2.0 module and its neighborhood. Through USB 2.0 module, we can transfer data, which comes from PC and will come to USB FIFO first, to DSP EVM. Also, USB 2.0 module can transmit data coming from DSP EVM to PC. We can connect PC with a web camera to generate video stream as our data source.

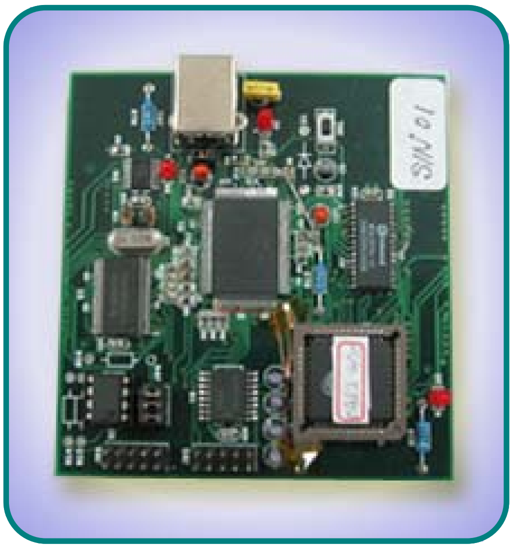


Figure 3.15: USB 2.0 module

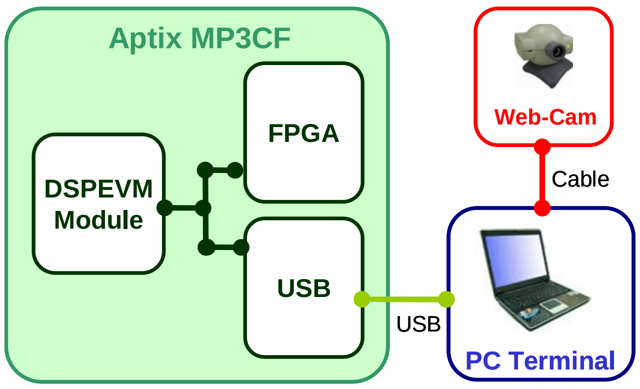


Figure 3.16: USB 2.0 module and its neighborhood

There are four transmission type supports by USB 2.0 module as follows:

1. **Bulk:** It guarantees the correctness of data transmission, but not the time spent. The typical application is used for burst data, such as PC sending printing data to a printer.
2. **Isochronous:** It guarantees the time spent, but not the correctness of data transmission. The typical application is used for transmission of image or voice files.
3. **Control:** It is used for the control of enumeration and device.
4. **Interrupt:** The typical application is used for expected poll time like the mouse event.

The USB 2.0 module uses only a 16-bit databus while the databus used in DSP EVM modules is 32-bit. Owing to the mismatch of databus, we need to assign the databus in USB 2.0 module to a 16-bit databus counting from the least significant bit (LSB) in DSP EVM, so that we can perform data transmission correctly.

In fact, USB 2.0 module provides an interface between PC and Aptix® MP3CF platform. In PC, we use the software called EZ USB Control Panel to control the file transmission between PC and USB 2.0 module. We use the endpoint 2 pipe to send data to USB 2.0 module; on the other hand, we use the endpoint 6 pipe to receive data from USB 2.0 module. Both of them are using Bulk type that has been mentioned above, as shown in Table 3.1.

Table 3.1: USB 2.0 module pipe mapping table

Endpoint	Type	Alternate Setting			Endpoint	Type	Alternate Setting		
		0	1	2			0	1	2
		Max Packet Size (bytes)					Max Packet Size (bytes)		
0	CTL	64	64	64	6 OUT	BULK	0	64	64
1 IN	INT	0	16	64	8 IN	ISO	0	16	256
2 IN	BULK	0	64	64	8 OUT	ISO	0	16	256
2 OUT	BULK	0	64	64	9 IN	ISO	0	16	16
4 IN	BULK	0	64	64	9 OUT	ISO	0	16	16
4 OUT	BULK	0	64	64	10 IN	ISO	0	16	16
6 IN	BULK	0	64	64	10 OUT	ISO	0	16	16

3.6 DAC and ADC Modules

In our adaptive 4x4 MIMO-OFDM system, we use the dedicated DAC and ADC modules to do the conversion between digital and analog signals as illustrated in Figure 3.17. The major components of each module include eight DAC/ADC chips, clock source, four databuses, and eight I/O ports, and are descript as follows.

1. **DAC/ADC Chips:** DAS825E and ADC900u are used as core chip respectively.
2. **Clock Source:** It can be setup by the combination of JP1, JP2, and JP10 jumpers.
3. **Databus:** Through the configuration of virtual pins in Aptix® Explorer, databus can receive and sent signals from and to FPGA modules by specific cables.

In addition, the output of DAC contains eight resistors numbered from *R219* to *R226*. When DAC is connected to ADC, we need to use 0.1 Ω resistors. But if we attempt to connect with the instrument that has 50 Ω input resistant, we must change resistors to 50 Ω to avoid the impedance mismatch problem, which will make signals decay.

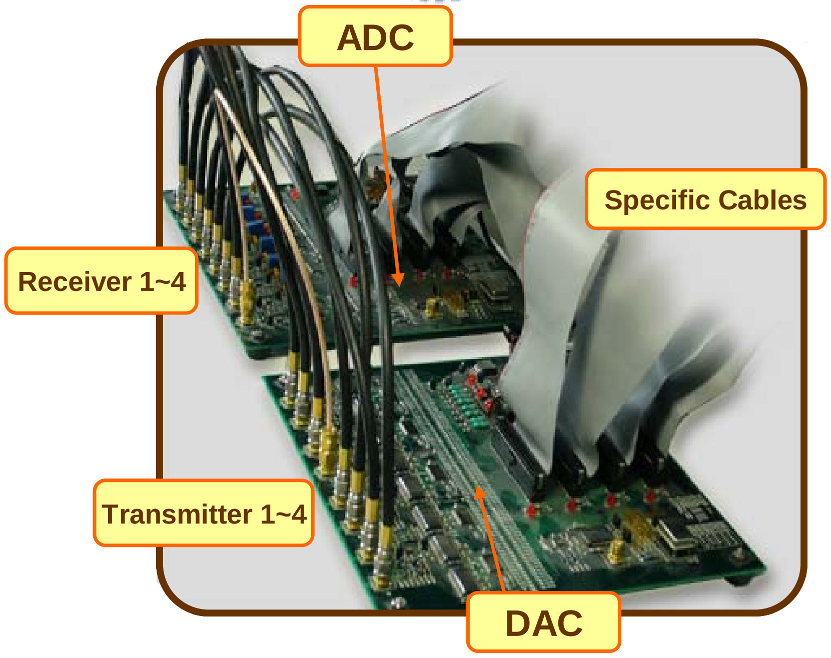


Figure 3.17: DAC and ADC modules

3.7 Debugging Tools

As an old saying goes, “What is a workman without his tools.” In our fast prototyping system, we do have some useful tools for debugging as follows.

1. **Logic Analyzer:** We use Agilent 16702B LA to perform the major task of debugging. There are two modules installed on it. One is 16522A Pattern Generator Module, and the other is 16711A Measurement Module. The former is mainly used for generating desired signals, such as the reset signal or some selection signals for model selection; the latter is used for probing signals in FPGA on Aptix® MP3CF platform by connecting specific pods to the slots on Aptix®.
2. **Oscilloscope:** It is usually used when transmitted signals are prepared by FPGA and sent to the DAC module by specific cables. Therefore, we can verify the waveform shown in the oscilloscope. For OFDM signals following IEEE 802.11a, we may expect to see the waveform containing preambles in the form of square wave in the head part and OFDM symbols follow behind those preambles.

3.8 Summary

In this chapter, we try to give a picture of the employed fast prototyping system. First, we introduce the Aptix® System Explorer platform. Then we give an overview on those modules installed on Aptix® platform, including FPGA module and ‘C6701 DSP EVM, and explain how they communicate with each other. In addition, USB2.0 module and ADC/DAC modules are also introduced. Further, we introduce the ADC and DAC modules which are in the form of dedicated boards. Finally, we give an overview of the debugging tools.