HOLT INC.
INTEGRATED CIRCUITS

HI-3210 ARINC 429
DATA MANAGEMENT ENGINE
OCTAL RECEIVER
QUAD TRANSMITTER

ADK-3210 Application Development Kit
Users Guide

June 2016

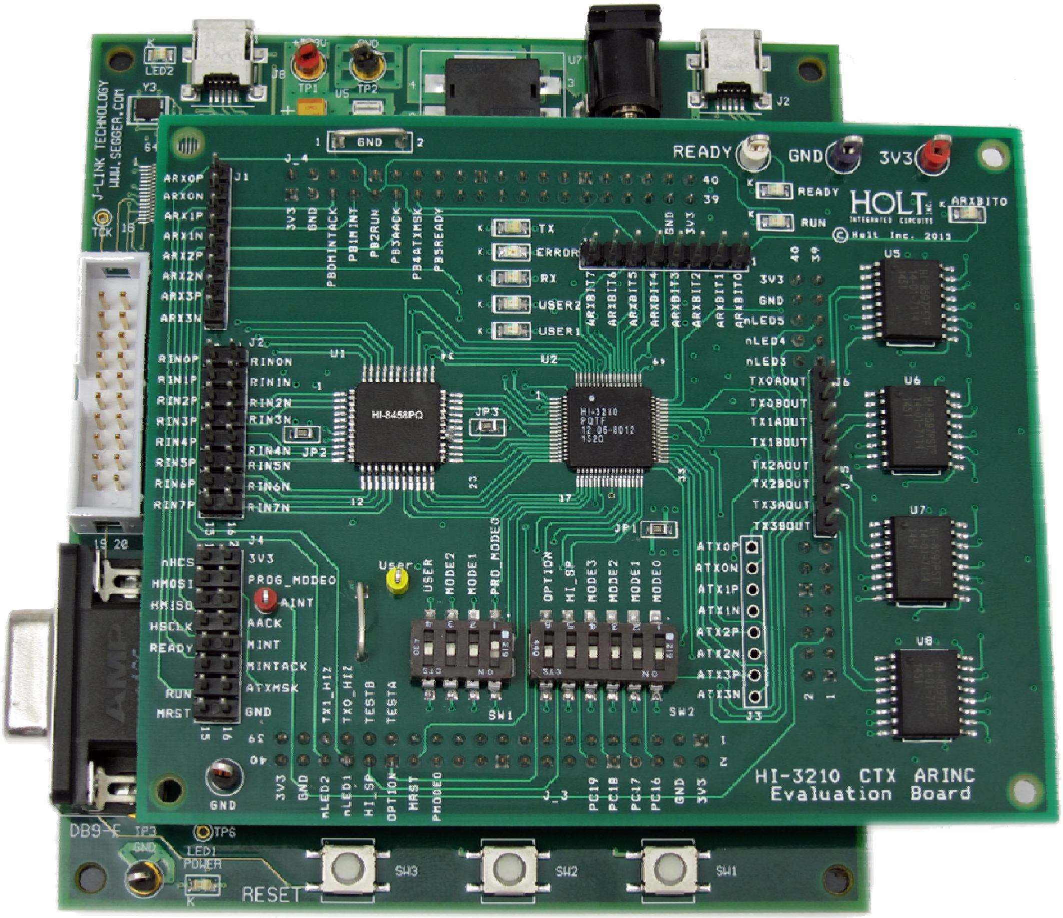
REVISION HISTORY

Revision	Date	Description of Change
AN-3210, Rev. New	06-21-16	Initial Release

Introduction

The Holt HI-3210 CTX Evaluation board demonstrates the broad feature set of Holt's ARINC 429 Data Management Engine, Octal Receiver and Quad Transmitter IC.

This guide describes how to set up and run the board. Additional support material and all required project software are found in the included Holt CD-ROM. A version of the demonstration software is pre-programmed into the microcontroller flash; the board is operational right out of the box without installing or running the provided software development tools.

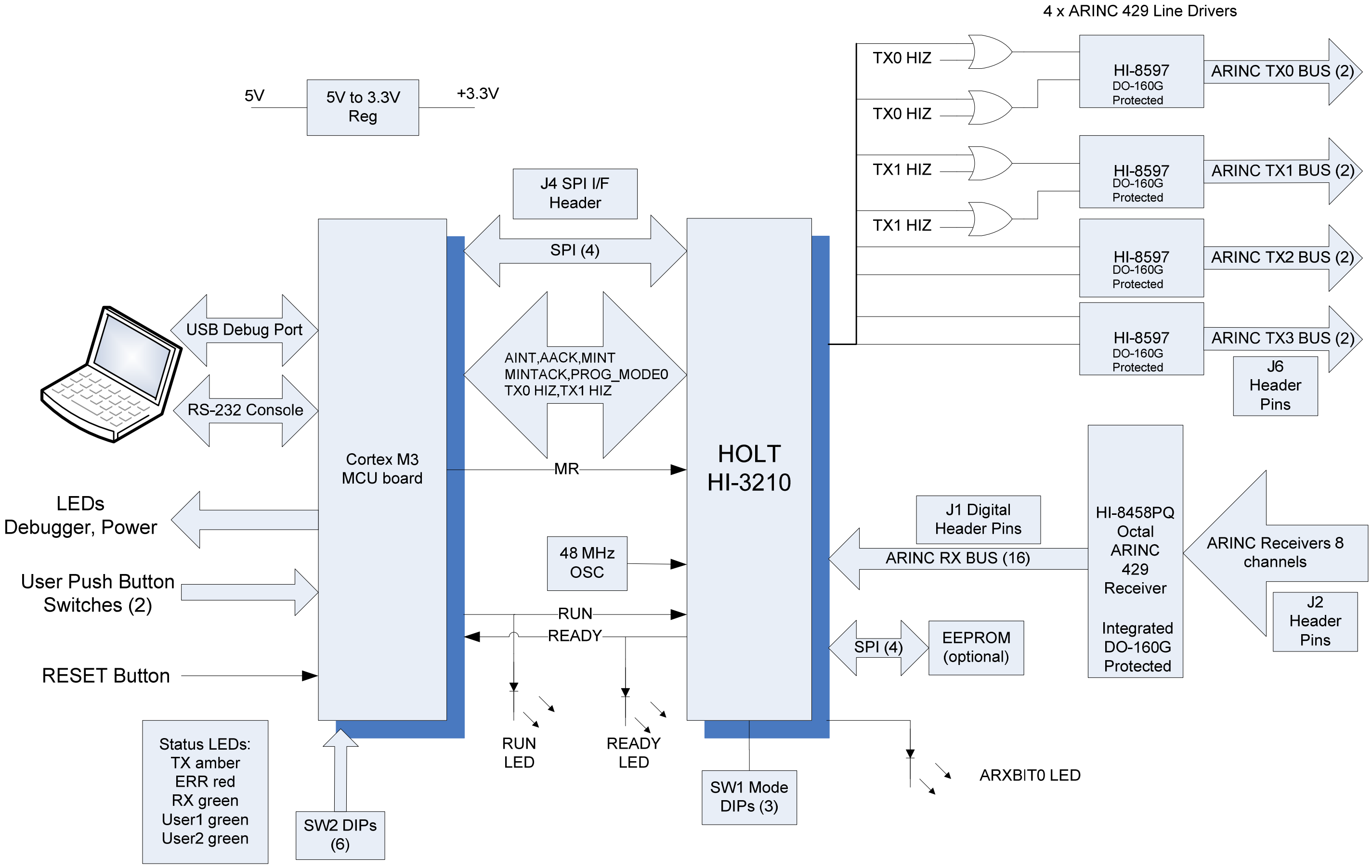


HI-3210 CTX Evaluation Board, mounted on the ARM Cortex M3 MCU Board

Evaluation Kit Contents

- This User Guide.
- Holt HI-3210 Project Software and Documentation CD.
- Installation CD for IAR Systems Embedded Workbench® for ARM (32KB KickStart.)
- Plug-in 5V DC power adapter.
- USB debug mini interface cable.
- RS-232 serial cable, DB-9M to DB-9F for console I/O using a connected computer.
- USB to serial adapter.
- 2-board assembly comprised of:
    - Upper HI-3210 evaluation board - top. Numerous DIP switches configure board operation.
    - Lower MCU board with ARM Cortex M3 16-/32-bit microprocessor, debug interface and regulated 3.3VDC power supply

Hardware Block Diagram



Default Switch Settings

3210 STARTUP CONFIGURATION

Switch	POSITION	DESCRIPTION
SW1, 4	Don't care	Optional pull-up resistor. No intended use.
SW1, 3:1	0,0,1	Set for Mode 1 demonstration purposes. PROG high, the 3210 auto initializes from EEPROM (set at Holt with demo 6 Repeater). Direct connection to 3210. MCU can drive the PROG pin when SW1-1 is open.

OPTIONAL DIP SWITCHES

SWITCH	DEFAULT	DESCRIPTION
SW2, 1	ON	MODE-0 DIP Switch. Available to user.
SW4, 2	ON	MODE-1 DIP Switch. Available to user.
SW4, 3	ON	MODE-2 DIP Switch. Available to user
SW4, 4	ON	MODE-3 DIP Switch. Available to user.
SW4, 5	ON	HI_SP. Reserved for future use. Available to user.
SW4, 6	ON	Option – Used to control Hi-Z mode on Line Drivers. OFF when the switch is on (0).

Default Jumper Settings

JUMPER	Solder pad	DESCRIPTION
JP1	IN	Supplies 3.3V to U2 HI-3210
JP3	IN	Supplies 3.3V to U1 HI-8458
JP3	IN	Supplies ground to U1 HI-8458

Getting Started

The first section of this users guide provides instructions how to set up the demo board and run the built-in demo programs and utilities pre-programmed into the Atmel Cortex M3 MCU. The second portion instructs how to install the IAR EWARM IDE and the Holt demonstration program. The supplied demo program on the CD-ROM is the same version already programmed in MCU flash memory.

Hardware Design Overview

Refer to the end of this guide for separate schematic diagrams and bills of material for the upper DUT (device under test) board and lower MCU board.

The detachable DUT board can be separated from the provided MCU board for connection to a user-supplied alternate microprocessor or FPGA board. The inter-board headers are located on 0.1” (2.54 mm) grid for compatibility with generic prototyping boards. All host interface signals go through the inter-board headers. Numerous configuration pins are controlled by DIP switches on the upper DUT board; these signals are not available on the inter-board headers.

The lower ARM Cortex M3 board is based on the flash-programmable Atmel AT91SAM3U-EK microprocessor. A 4-signal Serial Peripheral Interface (SPI) connects to the HI-3210 DUT. A UART-based serial port provides RS-232 console I/O (optional). An uncommitted USB 2.0 port is available for future expansion. Two pushbuttons are available for software interaction. A RESET pushbutton resets the ARM microprocessor, which in turn controls the DUT Master Reset signal.

The ARM Cortex M3 board includes “J-Link On Board” debug interface, licensed from www.segger.com, providing out-of-box readiness without having to buy a costly JTAG debug cable. The kit includes a simple USB cable for connecting the board’s debug interface to your computer. (For users already owning an ARM debug interface with ribbon-cable connector, an ARM-standard 2x10 debug connector provides debug connectivity. In this case, jumper JP2 on the bottom of the lower board should be soldered closed to disable “J-Link On Board”).

Initial Set Up

Before attempting to run the demo programs, a suitable PC is required with serial COM support so the demo programs and utilities can be invoked by entering commands on the console menu. These demo programs are already flashed into the Atmel Cortex M3 MCU and operate out of the box – no IDE flash programming is required for this.

1. Your PC will need a serial (COM) port and a “terminal emulation” program like TeraTerm. Most computers no longer have RS232 com ports so will require a serial-to-USB adapter, supplied with the ADK. Connect this to the computer’s USB port and the 9 pin connector to the ADK board.
2. If using Windows 2000 or Windows XP, you can use HyperTerminal for terminal emulation. Open HyperTerminal by clicking **Start** then **All Programs;** click the Windows **Accessories** then **Communications** program group. Double-click HyperTerminal to run it. Skip the next paragraph.

   If using Vista, Windows 7 or Windows 10…

   HyperTerminal is not included with these versions of Windows. Install the free open-source terminal emulation program, *TeraTerm 4.71*, by running the provided teraterm-4.71.exe installer program from the Holt CD. Accept the license agreement stating redistribution is permitted provided that copyright notice is retained. The notice can be displayed from the TeraTerm window by clicking **Help** then clicking **About TeraTerm**. Continuing to install…

   - Accept the default install destination and click **Next**.
   - At the Select Components screen, unselect all options except Additional Plug-in = TTXResizeMenu and click **Next**.
   - Select the installed language, then click **Next**.
   - Accept the default Start Menu folder, then click **Next**.
   - Select any desired shortcuts, then click **Next**.
   - At the Ready to Install screen, click **Install**.

   Run the TeraTerm program. At the **New Connection** screen, select **(x)Serial** and choose the selected COM port. Click **Setup** then **Serial Port** to open the serial port setup window. Choose these settings: Baud Rate: 115200, Data: 8 bits, Parity: none, Stop: 1 bit, Flow Control: none.

3. Plug-in the provided 5V DC power supply and connect the cable to the power input jack on the lower circuit board. If TeraTerm is running and configured correctly, the command menu below should appear in the console window. This menu appears whenever board power is applied, or the RESET pushbutton is pressed. After verifying correct TeraTerm communication with the evaluation board, the terminal set up can be saved by clicking **Setup** then **Save Setup**.

4. At power up, all the LED’s will briefly flash. The READY LED should remain ON and the RUN LED should be OFF and the green USER1 LED flashes at 1.2 Hz. The USER1 LED only flashes when the program is in the main idle loop.

Demonstration Programs and Utilities

It is highly recommended that you first review the HI-3210 data sheet to gain basic understanding of the device so you get the most benefit from the demos and utilities.

All the console commands use SPI Opcodes to read to write data in the HI-3210. The ‘r’ command does not use any SPI Opcodes, it sets the RUN GPIO output pin of the MCU High or Low which is connected to the RUN input pin of the HI-3210. When the RUN pin is high the green RUN status LED is ON. More about the RUN pin later.

Demonstrations and Examples

The demo programs are modeled by the Example diagrams shown in the data sheet on pages 3, 4 and 5. In the demo descriptions the documentation will typically refer to these methods shown in the 3rd column. The demos demonstrate one or two of these methods to transmit and receive data.

Example 1	On-chip RAM reception.	RAM Reception
Example 2	Data reception using on-chip FIFO’s.	FIFO Reception
Example 3	TX Data transmission direct from CPU. (Transmit directly by MCU)	Transmit Direct
Example 4	TX Data transmission on-chip schedulers. Requires Descriptor table(s)	Transmit scheduler(s)
Example 5	Autonomous Concentrator or Repeater. Requires Descriptor tables(s)	Concentrator or Repeater

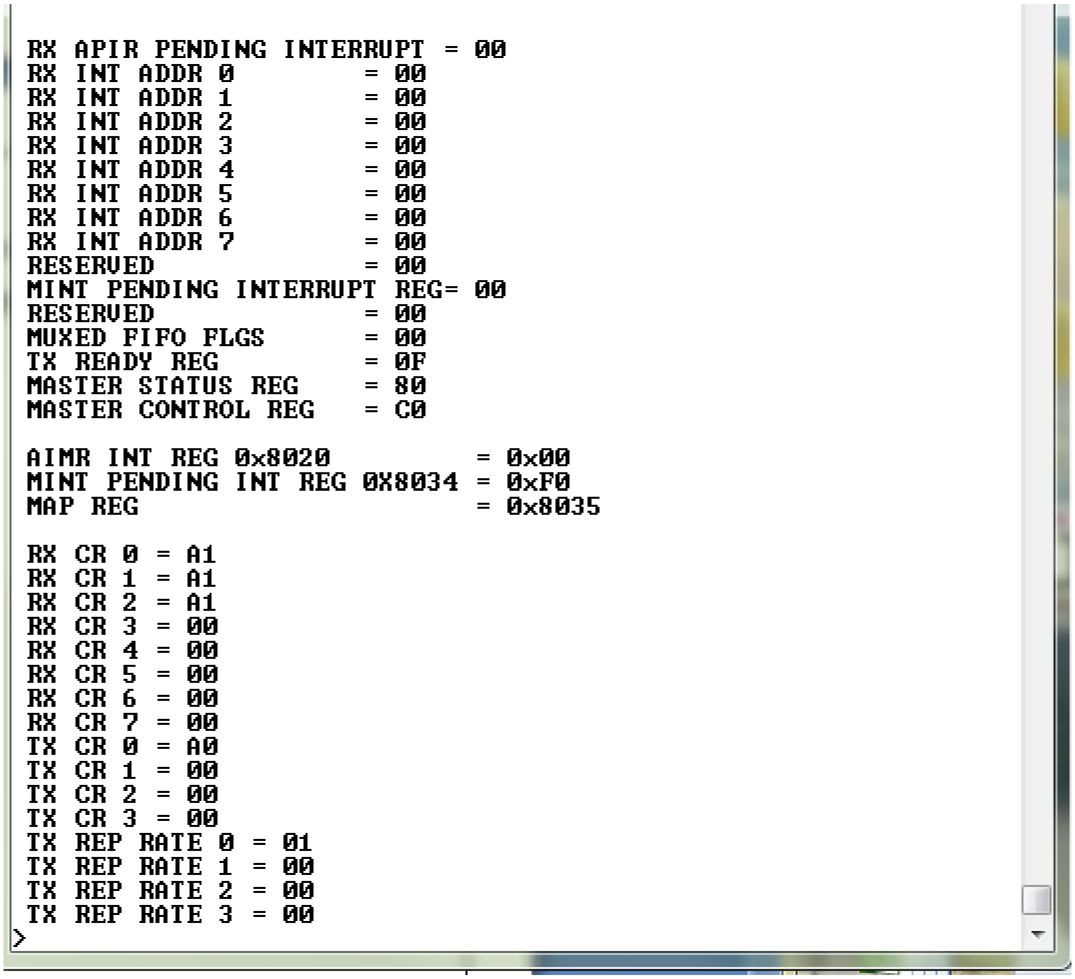
Utilities

SW1 and SW2 push button switches.

Two pushbutton switches on the bottom board provide two utility functions. Pressing SW1 also displays the selected set of registers like pressing the ‘Space Bar’. Pressing SW2 will prompt for a four digit hex address and use the watch window to display that block of device memory. These are only available from the main loop.

The OPTION DIP switch 6 by default should be closed. When the switch is detected open while executing the main.c loop, the MCU will assert both the TX0_HiZ and TX1_HiZ lines to two OR gates on U5 and U6 HI-8597 line drivers that puts the line drivers in Hi-Z mode. This allows connecting both U5 and U6 line driver outputs together as long as one line driver is in Hi-Z. HI-8597 line drivers maintain Hi-Z as long as they are powered.

Pressing the ‘Space Bar’ displays selected registers in the HI-3210. The first 16 registers are accessed using the “fast access” op codes to read the data. This utility is useful when learning and debugging new code. All displayed values are shown in hexadecimal with or without the “0x” prefix.



```
RX APIR PENDING INTERRUPT = 00
RX INT ADDR 0         = 00
RX INT ADDR 1         = 00
RX INT ADDR 2         = 00
RX INT ADDR 3         = 00
RX INT ADDR 4         = 00
RX INT ADDR 5         = 00
RX INT ADDR 6         = 00
RX INT ADDR 7         = 00
RESERVED              = 00
MINT PENDING INTERRUPT REG= 00
RESERVED              = 00
MUXED FIFO FLGS       = 00
TX READY REG          = 0F
MASTER STATUS REG     = 80
MASTER CONTROL REG    = C0

AIMR INT REG 0x8020         = 0x00
MINT PENDING INT REG 0X8034 = 0xF0
MAP REG                     = 0x8035

RX CR 0 = A1
RX CR 1 = A1
RX CR 2 = A1
RX CR 3 = 00
RX CR 4 = 00
RX CR 5 = 00
RX CR 6 = 00
RX CR 7 = 00
TX CR 0 = A0
TX CR 1 = 00
TX CR 2 = 00
TX CR 3 = 00
TX REP RATE 0 = 01
TX REP RATE 1 = 00
TX REP RATE 2 = 00
TX REP RATE 3 = 00
>
```

At power up or after a RESET, nearly all memory contains zeros.

Sometimes it is useful to clear all device memory from the console instead of resetting or power cycling the board. This guarantees fresh initialization values and device memory is cleared to zeros making it easier to identify updated values later. Press ‘c’ to clear all memory.

The ‘r’ command is used to toggle the state of the RUN pin on the HI-3210. This will become useful with the demos but for now press the ‘r’ key and see the RUN LED alternately turn ON and OFF.

Some console utilities are considered self-explanatory; these can be explored by the user.

The ‘a’ command prompts for a 4-digit hex address and displays that 256 byte region.

```
a
Type 4 hex char address 0000 through 7E00: 4000
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
 MAP:4000 40 51 A1 00 A2 00 A3 00 40 52 A5 00 A6 00 A7 00
 MAP:4010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:4090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:40F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
===============================================================
Keys: 'W' Watch   '<' Up   '>' Down   'R' Refresh   'A' Address
0x4000-0x40FF  End MAP:4100
================================================================
>
```

Command ‘w’ displays the 256 byte region beginning a location 0x0000. If the ‘a’ command was used, this address becomes the new staring address each time ‘w’ is pressed.

Command ‘L’ demonstrates two functions that set or clear ARINC Labels in the Received Data Enable Look-Up table shown on page 15 of the data sheet. Functions setLabel(chx,label) and clearLabel(chx,label) can be reused by the customer in their own C code projects. The input parameters for these functions are Receiver channel number (0-7) and label (0-255 dec). Setting the appropriate bit manually in the table is a tedious process otherwise.

Press ’L’ to set the labels called out in the demo.
Sets:
CH0 labels: 0, 16, 127 and 254.
CH7 labels: 1, 17, 128 and 255.

Results.
Use the ‘a’ or ‘t’ command to display this region of device memory. Notice the eight Labels are now enabled with the bits set. All other Labels which contain 0’s are disabled.

```
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
 MAP:7A00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 80
 MAP:7A10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40
 MAP:7A20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:7A30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:7A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:7A50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 MAP:7A60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
MAP:7A70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7A80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7A90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AE0 02 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00
MAP:7AF0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
```

The ‘m’ command just displays the MAP register contents.

Displaying HI-3210 memory and registers via SPI

Many of the utilities display selected areas in device memory. Some are fixed areas at specific start locations where the device stores ARINC 429 data, Interrupt enable tables and Transmit descriptor tables. The ‘w’ command is a general command that displays 256 bytes starting at location 0x0000. Pressing the ‘,’ < key subtracts 256 from the start address and ‘.’ > adds 256 to the start address. Press ‘a’ to enter a 4 digit start address from the console to view any desired memory block. The demo program has a rich selection of utilities for viewing device memory. This is important with the HI-3210 SPI interface since device memory cannot be viewed in a standard IDE debug memory window because the interface is SPI and not a parallel bus interface similar to SRAM. Commands ‘t’ and ‘d’ display the Transmit Schedule tables for TX0-TX3 and the full ARINC 429 Receive data areas of the part. See the HI-3210 memory MAP on page 7 of the data sheet. Some of these areas of memory are large so the displayed console data will scroll off the screen. When the display stops, simply scroll up to the desired address to examine device memory.

Demos

Most demos require external connections from the Transmitters to the Receivers (external loop-back) so that transmitted data is received by at least one Receiver so data is displayed on the console. This allows demonstrations without requiring external ARINC 429 test equipment. Demo '0' does not require external connections (jumpers or clip-leads) because it configures the HI-3210 for LOOPBACK mode. For this reason, Demo 0 is one of the easiest demos to use.

In some of the demos that continuously output data, pressing the 'q' key quits the demo and displays "Quit >" and returns control back to the console main loop. While executing some demos, hotkeys are available for changing certain parameters on-the-fly, for example to modify bits in the TX0 or RX0 control registers, begin data increment or stop the transmitter etc. In order to keep the demo code as simple as possible, only Transmitter 0 and Receiver 0 (TX0 /RX0) are altered by the hotkeys. Hotkeys are case sensitive.

Character	Hotkey Function
a	Modify Transmit Label and data (32-bits) value
e	Set RX0 and TX0 to Even Parity
E	Set RX0 and TX0 to Odd Parity (default)
i	Toggle Transmit data-increment ON/OFF.
p	Enable Parity on RX0 and TX0
P	Disable Parity on RX0 and TX0
s	Set low speed on RX0 and TX0
S	Set high speed on RX0 and TX0
x	Disable/Enable Transmission for all channels TX0-TX3

ARINC 429 Reception by RAM vs. Reception by FIFO (and host keeping up with data)

There are two methods to read ARINC 429 messages. Reception by RAM uses a single 4-byte buffer location mapped in 8K device memory according to channel number (0-7) and Label value. See the data sheet on page 13 for the memory organization. Reception by FIFO uses a 32 message FIFO for each Receiver channel (0-7).

Considerations as to which methods to use depend on user requirements, speed of host controller, software overhead and ARINC 429 traffic. When using Reception by RAM method, the host must read a message on the same channel and same Label before the next message arrives or the new message data will overwrite the previous data. If the next message is on a different channel or different label, no overwrite occurs because that channel or label maps into a different 4-byte buffer location. An AINT interrupt could be used to achieve fast response to reading these messages. In high speed, ARINC 429 messages could arrive about every 360µs.

Using FIFO Reception can help alleviate tight timing constraints since it stores up to 32 messages before potential overwrite when the host does not read messages fast enough to keep up with incoming messages. FIFO flags: FIFO Not Empty, Half Full and a FIFO Threshold register provide flexibility for designing a solution that processes messages fast enough to meet application requirements. A separate MINT interrupt pin is used with the FIFO. Because of the FIFO's ability to store up to 32 messages before overwrite data loss occurs, a polling strategy is often used instead of an interrupt.

The easiest way to configure the device to receive ARINC 429 messages is to use FIFO Reception method. The easiest way to configure and transmit ARINC 429 data is to use the MCU Direct Transmit method.

Another timing related topic is SPI communication latency. The HI-3210 maximum clock SPI rate is 20MHz. The Holt evaluation board is configured for 16MHz since MCU SPI uses a peripheral clock that is 96MHz/2 and SPI divide-by-3 provides 16MHz. A SPI divide-by-2 would generate a 24MHz SPI clock which exceeds the HI-3210 maximum SPI clock frequency, 20MHz. Here is a sequence in Demo 2 that reads the "FIFO Not Empty" register to determine whether or not to read FIFO data.

Set MAP register to FFNE location	Opcode + 2 bytes = 24 clocks
Read memory byte from MAP address	Opcode + 1 byte (FFNE reg value) 16 clocks
Opcode to read FIFO Not Empty FFNE register	Opcode + 4 bytes = 40 clocks

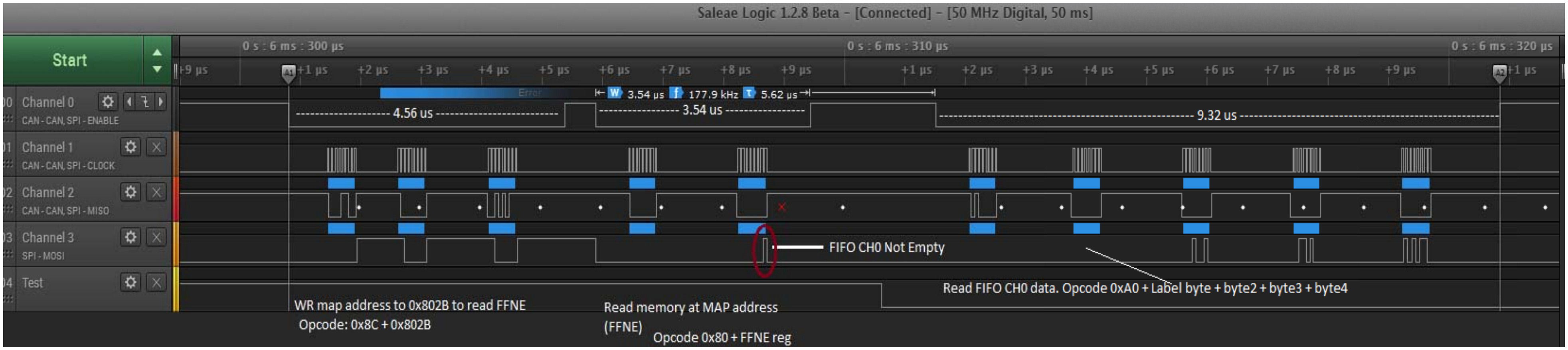
The total time to perform this sequence is the total number of clocks plus implementation delays of the MCU SPI master and software overhead.

Total clocking time = 80 * 1/16MHz = 5µs. This does not account for additional delays.

The following logic analyzer plot for this sequence shows the timing. It reveals various delays which are dominated by calls to low level driver functions including a function call to turn ON an LED which is partly responsible for the 2µs gap before reading the FIFO data. It's possible to optimize this by using inline code instead of function calls. This timing is still acceptable for most applications. Once the program detects presence of FIFO data, reading the message takes 9.32µs.

The total time for this example is about 20µs. In an FPGA implementation nearly all the idle gaps could be removed with some effort. See the timing diagram on the next page. Due to zoom level, the diagram is not very legible when printed, but on a computer screen it's quite viewable.

The SPI timing diagram also shows proper SPI sequencing to properly communicate to the HI-3210 with op codes and data. This is a live capture using the Saleae Logic analyzer using Demo 2. The yellow trace at the bottom was the RX LED turning ON (asserted low) and was used to trigger the logic analyzer.



FIFO CH0 Not Empty

Read FIFO CH0 data. Opcode 0xA0 + Label byte + byte2 + byte3 + byte4

WR map address to 0x802B to read FFNE
Opcode: 0x8C + 0x802B

Read memory at MAP address
(FFNE)
Opcode 0x80 + FFNE reg

Demo 0:

Press the ‘0’ key for RAM Reception - LOOPBACK.
Demo 0 programs the LOOPBACK register with 0xFF so all channels are configured in LOOPBACK mode. All four transmitters are configured in high-speed and Parity ON. All eight Receivers are similarly configured. Initially the transmit data is set for Label 01 and data 23 45 67. ARINC 429 data is transmitted to the four channels using the Transmit Direct method using Opcodes (0x94-0x97).

The output should repeatedly output this data.

```
RX-CH0: Label:01 StatusByte:3F Data:23 45 67
RX-CH1: Label:01 StatusByte:3F Data:23 45 67
RX-CH2: Label:01 StatusByte:3F Data:23 45 67
RX-CH3: Label:01 StatusByte:3F Data:23 45 67
RX-CH4: Label:01 StatusByte:3F Data:23 45 67
RX-CH5: Label:01 StatusByte:3F Data:23 45 67
RX-CH6: Label:01 StatusByte:3F Data:23 45 67
RX-CH7: Label:01 StatusByte:3F Data:23 45 67
```

While running this demo, press the ‘i’ hotkey. This causes transmit data to increment after a brief message “TX Auto_Increment ON”. Each Transmitter is now loaded with a unique value and increment occurs after each transmission.

Press ‘q’ to examine some of the data. It might appear similar to this. The data from each Transmitter is looped back to two Receivers according the table in the data sheet on page 18.

```
RX-CH7: Label:01 StatusByte:3F Data:23 45 67
TX Auto_Increment ON

RX-CH0: Label:01 StatusByte:3F Data:23 45 67
RX-CH1: Label:01 StatusByte:3F Data:23 45 67
RX-CH2: Label:01 StatusByte:3F Data:24 45 67
RX-CH3: Label:01 StatusByte:3F Data:24 45 67
RX-CH4: Label:01 StatusByte:3F Data:25 45 67
RX-CH5: Label:01 StatusByte:3F Data:25 45 67
RX-CH6: Label:01 StatusByte:3F Data:26 45 67
RX-CH7: Label:01 StatusByte:3F Data:26 45 67
```

The received data is read by RAM Reception by first polling the APIR (Interrupt pending register). When a channel is detected by examining the APIR register, the AIARx register is read to fetch the label address of the message and the exact address of the message is determined by RXaddr = blk+(4*rxAdd). The four bytes are read using the MAP register set to this address.

Optionally, instead of reading the Receiver data by RAM Reception, the data could have been read by FIFO Reception. This is demonstrated in Demo 2.

By default, the program is compiled for Polling using the macro INTERRUPT_MESG_ENA. A simple interrupt method to read the message is provided by changing this macro to a 1 in the 3210_initialization.h file.

While transmitting data the amber TX LED briefly turns On. When the demo is reading data from the receiver memory the green RX LED briefly turns On. The LEDs are controlled by the MCU before calling the functions to transmit or receive the SPI data then is turned OFF typically after a 1ms delay. The brightness varies depending on the amount of messaging and whether ARINC is operating at low or high speed.

All hotkeys are handled in function hotkeys() in the 3210Demo.c module.

Demo 1:
Press the '1' key for RAM Reception - normal mode.

Demo 1 is the same demo as Demo 0 except the LOOPBACK feature is turned OFF (LOOPBACK register 0x8022 = 0x00). This demo transmits data on the four transmitters. An oscilloscope can be used to view the ARINC transmissions on J6 header pins. For example pin 1 is TX0AOUT and pin 2 is TX0BOUT. Without jumper wires connected between a Transmitter and a Receiver, no console data is displayed.

The eight Receive input pairs are on J2 header pins. Since all transmitters and all receivers are active in this demo, you can connect any number of receivers to a transmitter. Nearly any configuration is possible, except transmitter outputs should not be connected together unless one transmitter is in Hi-Z mode.

Connect TX0 output pairs to RX0 input pairs and verify that data is displayed on the console when running the demo.

J6-1 TXOAOUT	Jumper to	J2-1 RINOP
J6-2 TXOBOUT	Jumper to	J2-2 RINON

```
ARINC RAM Reception
Normal Mode

RX-CH0: Label:01 StatusByte:3F Data:23 45 67
RX-CH0: Label:01 StatusByte:3F Data:23 45 67
RX-CH0: Label:01 StatusByte:3F Data:23 45 67
```

To see data increment, press the 'i' hotkey while running. Increment does not alter the label value.

To see this same data on Receiver channel 1, change the two jumpers on J2 to J2-3 and J2-4 respectively.

```
RX-CH1: Label:01 StatusByte:3F Data:9B 23 68
RX-CH1: Label:01 StatusByte:3F Data:9F 23 68
RX-CH1: Label:01 StatusByte:3F Data:A3 23 68
```

Hotkey 'x' alternates between disabling and re-enabling the transmissions on all channels TX0-TX3. When the transmitters are disabled, the TX LED is off.

Hotkey 'a' allows modifying the transmit word (32-bits) by first prompting for eight hex digits. For example if 12 34 56 78 is entered the message would be label=12, data = 34, 56 and 78.

When using the Receivers with an external ARINC transmitting source remove any jumpers from the on-board transmitters on J6 and connect the external ARINC transmit signals on any Receiver pair on J2.

Demo 2:
Press the '2' key for TX Direct Transmit and FIFO Reception.

This demonstrates FIFO reception on all eight Receivers. This transmits data on channel TX0-TX3 so that there's transmit data that can be connected to the Receivers for demo purposes.

Each Receiver channel is supported by a 32-message FIFO. The Receiver Control registers are configured with FFS1=0 and FFS0=1 which sets the FLAG when the FIFO is not empty. If the IMR register bit-3 (FLAGIE) is set, then the MINT output pin will be asserted high when a message is received. Polling the FFNE register is used by this demo. All messages are read out of each channel until empty starting with Receiver 0 though to Receiver 7. See Demo 2 code section.

The Receivers and Transmitter are configured with mixed high speeds and low speeds.

High Speed:	TX0, TX1, RX0-RX3
Low Speed:	TX1, TX2, RX4-RX7

When adding jumper clips between transmitter outputs and receiver inputs, make sure the speed is matched or there will be no Receiver reception.

High speed: TX0-TX1, RX0-RX3
Low speed: TX2-TX3, RX4-RX7

The easiest way to read ARINC Receiver data is to read the FIFOs using the special op codes 0xA0-0xBC. There are eight RX channels, so each channel has a unique op code, 0xA0 to 0xBC. Reading

FIFOs without first checking for a new message in the FIFO (using the FIFO flags or interrupt registers) typically returns stale data.

This time, use Receiver 7 to capture the data from TX2 (both low speed) so connect the jumpers as shown:

J6-5 TX2AOUT	Jumper to	J2-15 RIN7P
J6-6 TX2BOUT	Jumper to	J2-16 RIN7N

To identify Transmit channel data when received, the label value for each Transmitter is unique. Transmitter Label values: TX0 = 00, TX1=01, TX2=02 and TX3=3.

The console output will appear like this with the auto increment OFF.

```
       CH   L B2 B2 B3 (Channel#, Label, Byte2, Byte3, Byte4)
RXFIFO 7: 02 12 34 56
RXFIFO 7: 02 12 34 56
RXFIFO 7: 02 12 34 56
```

Transmit Data Management Engine demos

Demos 3, 4, 5, 6, and 7 use the Transmit Data Management Engine. Separate descriptor tables, one for each Transmitter, contains up to 256 message descriptors. Each descriptor (per message) requires 4 action bytes and 4 value bytes. Refer to the data sheet section ARINC 429 Transmit Scheduler for the full description. Once the descriptors are initialized and Control Register bits are set, the scheduler runs when the RUN pin is set high. To stop the scheduler set the RUN pin low. Using the ‘r’ command is used to toggle the RUN pin in these demos. Using the Transmit Data Management Engine is a powerful feature in the HI-3210, but it is more complex to setup and use.

Demo 3:
Press ‘3’ for TX0 Simple Scheduler

This demo initializes the Transmit channel TX0 descriptor table with 16 messages using the Immediate op codes for the action values. All 16 messages are the same, Label 00, data: 0x82, 0x83 and 0x84.
After each message is loaded in the descriptor table, the label increments. After the program loads all 16 descriptors, an End of Sequence op code is written in the next address location with 0x00.
The repetition rate is set for 40ms.

The demo quickly returns to the main console loop so there’s no hot key support.
Press ‘r’ to set the RUN pin high to start the scheduler, the 16 messages will transmit every 40ms. If ‘r’ is pressed again, RUN goes low and transmission stops.

View the ARINC TX channel 0 transmissions on J2 pin 1 and 2 with an oscilloscope.

To read back the data and display it on the console using the FIFO Reception method, first connect the TX0 transmitter outputs to the RX0 Receiver inputs as follows.

J6-1 TXOAOUT	Jumper to	J2-1 RINOP
J6-2 TXOBOUT	Jumper to	J2-2 RINON

Rerun the demo and press ‘r’ to transmit the data, then press ‘r’ again to stop transmission. Press ‘9’ to display 32 messages on the console. There are 32 messages because that is the FIFO depth. Probably several 16-message groups were transmitted between the first and second ‘r’ key presses, and the FIFO is full because there were no FIFO reads up to this point.

```
RXFIFO 0: 00 82 83 04 … RXFIFO 0: 0F 82 83 04  (first group of 16 messages)

RXFIFO 0: 00 82 83 04 … RXFIFO 0: 0F 82 83 04  (second group of 16 messages-same data)
```

The MSB of the last byte is the parity bit and although it was set high in the descriptor table it is received as a low because Odd parity is enabled.

You can also leave RUN high and randomly press ‘9’ to display the FIFO messages. Typically a random amount of messages (typically greater than 32) are displayed because it’s a race between how fast the FIFOs fill with message data vs. how fast the console can read them out. The display rate is bound by the serial 115200 baud rate and host PC.

Demo 4:
Press ‘4’ for TX0 Simple Scheduler. One-time Sequence transmission method 1.

This demo uses the same exact demo code function as Demo 3, except a zero value is used in the rate parameter. This demonstrates One-Time sequence transmission using Method 1 described in the data sheet on page 22.

Press ‘4’ then Press ‘r’.
Only one set of 16 messages will transmit. If jumpers are still in place from Demo 3, then Press ‘9’ to see the 16 messages. Attempting to Press ‘9’ again will show nothing. Only another low-to-high transition on the RUN pin (using the ‘r’ key) will transmit the sequence again.

Demo 5:
Press ‘5’ for TX0 Simple Scheduler. One-Time Sequence transmission Method 2.

This demo uses nearly the same demo code function as Demo 4 (with a zero value in the rate parameter) but uses Method 2 described in the data sheet to perform the One-Time sequence which is accomplished by cycling the state of the RUN pin low-to-high.

Demo 6:
Press ‘6’ for RX0 -> TX0 Repeater.

This demo sets up a single descriptor in the TX0 scheduler table to perform a Repeater function and returns control back to the console main loop. Pressing ‘r’ sets the RUN pin high to start the scheduler. Any message received on Receiver 0 automatically transmits on TX0 without host intervention. Optionally, the host can receive the message if desired, using RAM Reception or FIFO Reception.

The descriptor is configured to reference RX0 channel, label 0 message in RAM memory (0x0000).

There are six scheduler op codes available shown on page 23 of the data sheet. This example uses an Immediate Opcode for byte-1 (Label) so that the retransmitted message will have a different label 0x5A. Bytes 2, 3 and 4 use Indexed data conditional op codes so these three bytes are fetched from the incoming message referenced by the RX0 channel and label address. Review Demo 6 function code in 3210Demo.c to see how this is configured.

In the code the ‘action’ and ‘value’ pairs are configured in two arrays before writing the data to the descriptor table.

```
act[4] = {
        IM,                         (immediate Opcode)
        IDC | CH0_CCC | BB2,        (Indexed conditional Opcode referencing Byte 2)
        IDC | CH0_CCC | BB3,        (Indexed conditional Opcode referencing Byte 3)
        IDC | CH0_CCC | BB4}        (Indexed conditional Opcode referencing Byte 4)

val[4] = { 0x5A,  (because act=IM value 0x5A is transmitted as the label)
           0,     (because act is IDC opcode, RX ch 0 label 0 message byte2 is transmitted)
           0,     (because act is IDC opcode, RX ch 0 label 0 message byte3 is transmitted)
           0}     (because act is IDC opcode, RX ch 0 label 0 message byte4 is transmitted)
```

To view the descriptor action and value byte pairs for this descriptor in device memory, use the ‘a’ command and enter address 4000. The descriptor memory should appear with the following data (only partially shown):

```
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
MAP:4000 40 5A A1 00 A2 00 A3 00 00 00 00 00 00 00 00 00
                                 ^--- EOS
```

For example:

RX0 Incoming message: 00 02 34 56
Repeated message to: 5A 02 34 56 (repeated message)

For demonstration purposes, make an external connection between TX2 outputs and RX0. This configures the TX2 transmitter to be a message source for the RX0 input channel.

Steps:
Press ‘7’ to run demo.
Press ‘r’ to run scheduler (RUN high)
Press ‘k’ to transmit test messages on TX2 and TX3 (TX3 is ignored.)
Press ‘9’ to display the messages.

To see the repeated message transmission, connect a pair of jumper leads from the TX0 output pins to the RX3 channel inputs. Only RX0-RX3 Receivers are initialized in this demo.

Results:

```
> 6
Repeater
Press r to set RUN high to run scheduler
Press k to transmit TX2 TX3 test messages
Press 9 to read the FIFO data and display it on the console
> r
> k
Transmit Two direct Test messages on TX2 and TX3
> 9
RXFIFO 0: 00 02 34 56
RXFIFO 3: 5A 02 34 56 >
```

There is a 10ms latency delay between the received message and the transmitted message. This is due to the way the scheduler works. When the RUN pin is set high, the scheduler starts at sequence 0 driven by the HI-3210 internal 10ms repetition rate timer; the timer is asynchronous with received ARINC messages. This demo has one sequence. When it finishes sequence 0, it repeats sequence 0 again after every 10ms. The minimum repetition time is 10ms.

Indexed Conditional op codes use NEWTX0 – NEWTX3 bits in the Received Data Status Bytes. This Status Byte is the first byte in Received data memory (See page 13 of the data sheet) that’s loaded for every received message. When a message is received all the bits in this Status Byte register are set. This is how the scheduler detects when a new message is received. If a descriptor references any of the data from this block, the corresponding TXx bit is reset depending on the Transmitter channel (TX0-TX3).

Demo 7:
Press ‘7’ for Concentrator Demo.

The concentrator demo receives messages on two Receiver channels (RX0 and RX1) and retransmits them both on TX0 transmitter.

This demo sets up two descriptors in the TX0 scheduler table. The first descriptor references the RX0 channel message and the second one references the RX1 message (both Label 0). After pressing ‘7’, control returns back to the main console loop. Press ‘r’ to set the RUN pin high to start the scheduler. Any new messages received on Receiver 0 or Receiver 1 will automatically re-transmit to TX0 with no host intervention. Optionally, the host can receive the messages if desired, using either RAM Reception or FIFO Reception.

Eight byte arrays are used for this demo to contain the two descriptors. Using an array is not required to load descriptor memory; it’s useful here for instructional purposes.

The arrays in demo function Concentrator() in module “3210Demo.c” are shown below with additional comments.

```
act[8] = {
        IM,                         (immediate Opcode)
        IDC | CH0_CCC | BB2,        (Indexed conditional Opcode referencing Byte 2)
        IDC | CH0_CCC | BB3,        (Indexed conditional Opcode referencing Byte 3)
        IDC | CH0_CCC | BB4,        (Indexed conditional Opcode referencing Byte 4)
        IM,                         (immediate Opcode)
        IDC | CH1_CCC | BB2,        (Indexed conditional Opcode referencing Byte 2)
        IDC | CH1_CCC | BB3,        (Indexed conditional Opcode referencing Byte 3)
        IDC | CH1_CCC | BB4}        (Indexed conditional Opcode referencing Byte 4)

val[8] = {
        0x5A,   (because act=IM value 0x5A is transmitted as the label)
         0,     (because act is IDC opcode, RX ch 0 label 0 message byte2 is transmitted)
         0,     (because act is IDC opcode, RX ch 0 label 0 message byte3 is transmitted)
         0,     (because act is IDC opcode, RX ch 0 label 0 message byte4 is transmitted)
        0x5A,   (because act=IM value 0x5A is transmitted as the label)
         0,     (because act is IDC opcode, RX ch 1 label 0 message byte2 is transmitted)
         0,     (because act is IDC opcode, RX ch 1 label 0 message byte3 is transmitted)
         0}     (because act is IDC opcode, RX ch 1 label 0 message byte4 is transmitted)
```

View the descriptor table at location 0x4000 (using the ‘a’ or ‘t’ command):

```
                 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
MAP:4000 40 51 A1 00 A2 00 A3 00 40 52 A5 00 A6 00 A7 00
MAP:4010 00 (this byte is EOS)
```

In order to best demonstrate this demo, command ‘k’ will be pressed after pressing ‘7’. Command ‘k’ transmits a test message on TX2 and TX3. TX2 and TX3 outputs should be jumpered to RX0 and RX1. And one final connection (so we can see the result on the console) is to jumper TX0 to RX3.

TX2 will transmit 00 02 34 56 → Received by RX0

TX3 will transmit 00 03 34 56 → Received by RX1

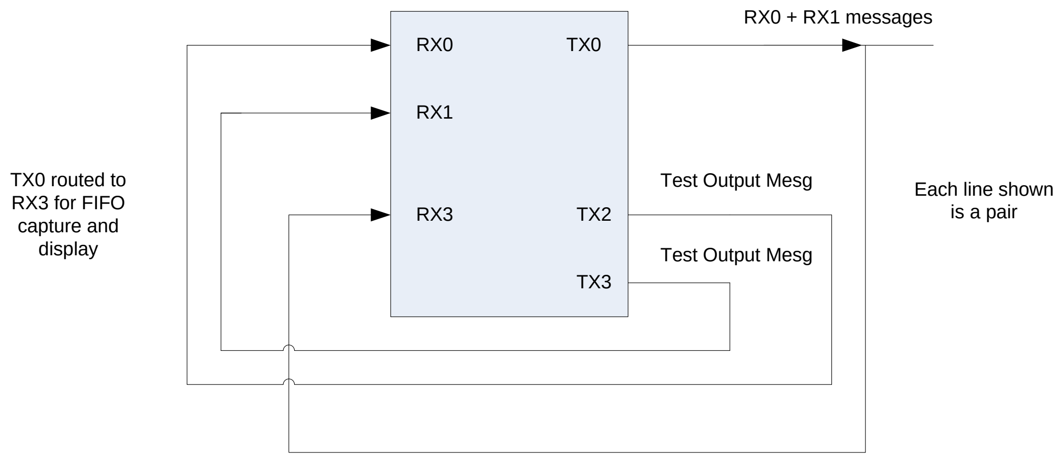
TX0 (from the scheduler) transmits two messages 51 02 34 56 and 52 03 34 56.

Results:

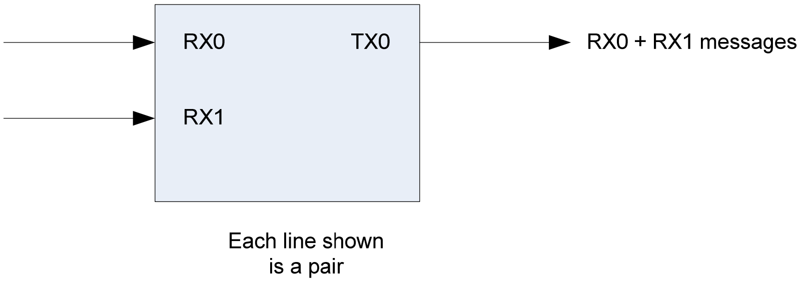
```
> 7
Concentrator Demo
Press r to set RUN high to run scheduler
Press k to transmit TX2 TX3 test messages
Press 9 to read the FIFO data and display it on the console
> r
> k
Transmit Two direct Test messages on TX2 and TX3
> 9

RXFIFO 0: 00 02 34 56
RXFIFO 1: 00 03 34 56
RXFIFO 3: 51 02 34 56
RXFIFO 3: 52 03 34 56 >
```

A block diagram shows the connections for demoing purposes.


RX0 + RX1 messages
RX0
TX0
RX1
TX0 routed to RX3 for FIFO capture and display
Test Output Mesg
Each line shown is a pair
RX3
TX2
Test Output Mesg
TX3

In an optimized two channel input to one channel output Concentrator is shown without all the loop-backed connections.



As was shown in the demo diagram, connecting outputs back into the HI-3210 inputs can provide some additional functionality in some creative ways. One example of this is using one transmit channel to transmit messages at one interval, while using a different repetition rate on another transmit channel and looping that back into a spare input. The output can combine both sets of messages depending on the how the descriptors are programmed in table memory. If this technique is used in a production design, the Line Driver and Receiver and be eliminated. Connect the digital TXx outputs directly to the digital inputs to the HI-3210.

Demo p:
Press ‘p’ to program the auto-initialization EEPROM.

Using the HI-3210 with an EEPROM allows it to auto-initialize and operate independently of a MCU or FPGA. Once the configuration data is stored in the EEPROM, it will automatically load at RESET if the Mode [2:0] inputs are configured for Mode 1 or Mode 3 according to the flow chart on page 27 of the data sheet. Demo ‘p’ initializes the EEPROM with the Demo 6 Repeater demo.

Demo steps:

1. Close Mode2 and Mode1 SW1 DIP switches (down position), Open PROG_MODE0 DIP SW1 DIP switch. When PROG_MODE0 is open it allows the MCU to control the state of the pin. This pin is configured as an open-drain output of the ARM Cortex MCU.
2. Press ‘6’ to run the Repeater demo to initialize the HI-3210. Demo 6 is operational at this point.
3. Press ‘p’ to program the EEPROM. The ERROR Red LED briefly flashes.
4. Power-cycle the board or Press RESET. The HI-3210 should auto-initialize from EEPROM so Demo 6 will operate without having to Press ‘6’ from the console menu. Refer to Demo 6 for demonstration instructions.

Auto-initialization from EEPROM is an optional feature; an EEPROM is not needed if this feature is not used.

Final Tips:

1. The RUN input must be low to write to RX Control registers.
2. Terminate the last descriptor table written in device table memory with an EOS (0x00) byte.
3. When reading or writing a device memory location other than 0x8000 – 0x800F, the Memory Address Pointer (MAP) register must first be written with the desired read or write address before accessing memory using read or write op codes 0x80 or 0x84.

Getting Started with the Holt API demo software project and installing IAR Systems *Embedded Workbench for ARM* Compiler

1. Installed IAR Systems *Embedded Workbench for ARM* (*EWARM*) compiler is required BEFORE adding the Holt demo project, so all Atmel board library files and the demo project folder are created in the proper locations. Follow *the "Holt HI-3210 Demo Project Installation Guide"* found in the Project folder on the Holt CD-ROM. Before proceeding to the next steps IAR must be installed and the Holt project folder must be in the proper folder location, according to that guide. **Instructions beyond this point assume you have completed the above installation tasks.**

2. Launch IAR *Embedded Workbench* from the Windows Start menu. A blank screen should appear. Open the Holt HI-3210 Demo Project from the IAR File pull-down menu, click on File/Open/Workspace and navigate to the project folder location and select "HI-3210 Demo.eww" and click the Open button.

3. Debug requires an interface between the computer running IAR Embedded Workbench® and the HI-3210 Application Development Kit. Connect the small end of the provided USB cable to the evaluation board USB connector marked DEBUG. Connect the other end of the USB cable to a free computer USB port. The IAR C-SPY Debugger for ARM includes drivers for numerous target system interfaces, including built in "J-link On Board".

   The first time the evaluation board USB cable is connected to the computer, the Windows "Found New Hardware" message should appear for the J-Link device. After several seconds, Windows should load the appropriate driver and advise, "Your hardware is ready for use". If Windows fails to find the J-Link driver, direct it to look in the Drivers directory the IAR Embedded Workbench® installation CD.

   If difficulties arise when initiating a debug session at step 5, click **Project** then **Options**. In the window that opens, under **Category** = **Debugger** highlight **J-Link/J-Trace**. Click the tab labeled **Connection**, then verify Communications = USB and Interface = SWD.

3. Open IAR Embedded Workbench®. Click **File**, then **Open Workspace**, then navigate to the project subdirectory created in step 4. Select the project file with .eww extension, then click **Open**. (The next time Embedded Workbench® opens, this project will appear in the Recent Workspaces list when **File** is clicked.)

4. If problems occur with IAR installation or with using the IAR debugger, two Holt technical notes are provided to help resolve these issues included on the Holt CD ROM.

5. The demo project only uses unsigned integer variables. Optionally turn off the nuisance compiler message that occurs when a variable's most significant bit toggles. The message looks like this:

   `Remark[Pe068]: integer conversion resulted in a change of sign`

   To disable this diagnostic message, click **Project** then click **Options**

   Category = C/C++ Compiler

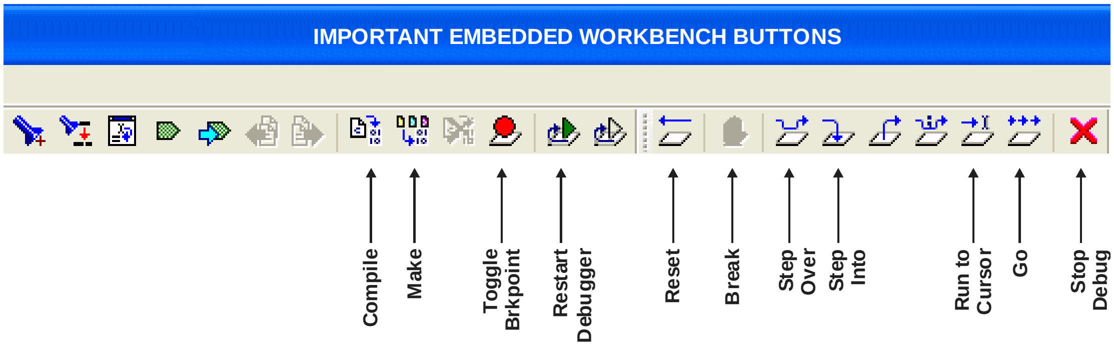
   Tab = Diagnostics

   `Suppress these diagnostics:` add "Pe068" to list

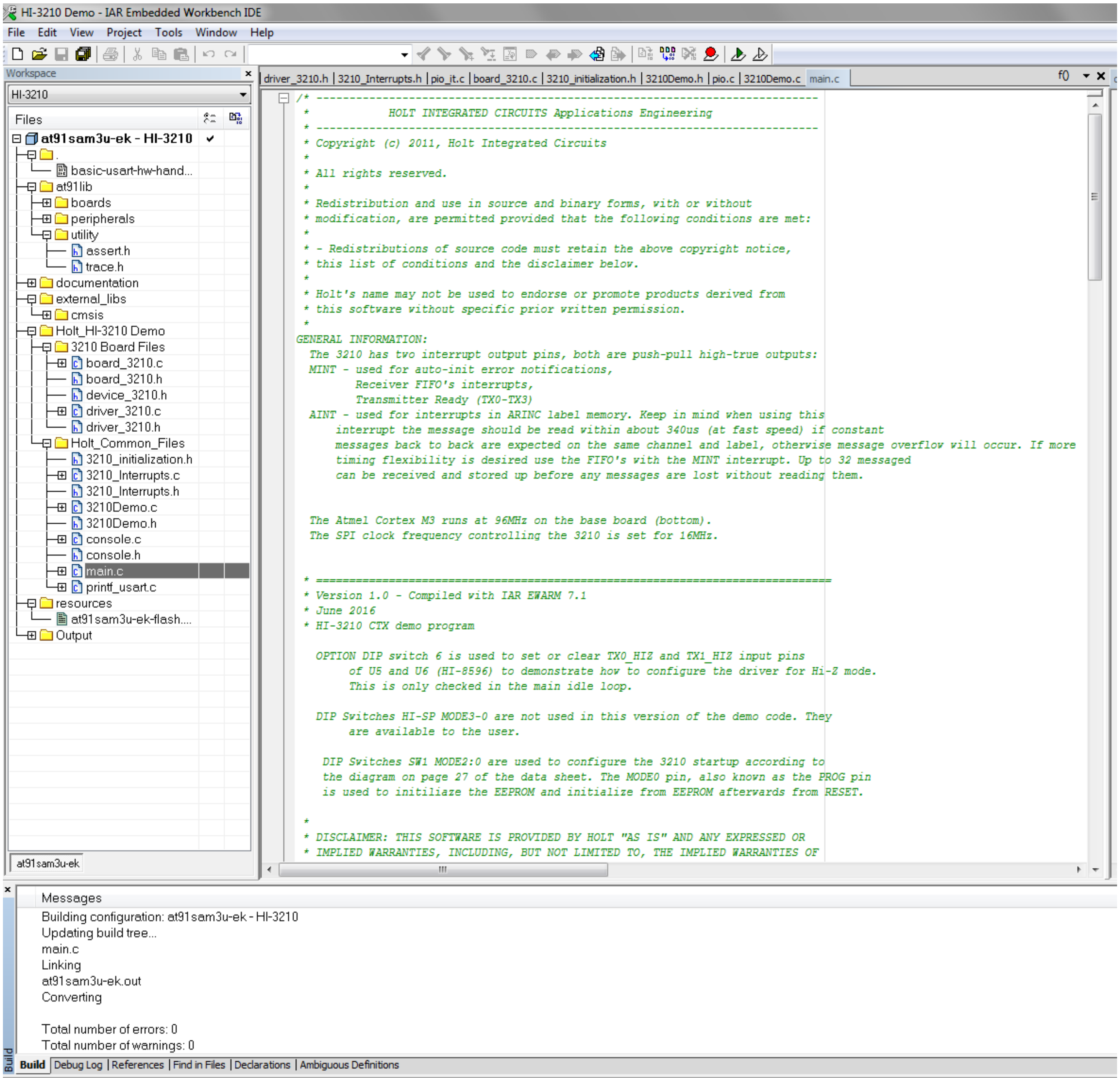
6. RAM based projects are not supported due to the limited amount of RAM on the MCU. By design the Cortex™-M3 runs slower in RAM than in Flash so there is little need for a RAM based project.

   Compile the project by clicking the **Make** button. See following illustration. If the Build messages window in IAR Embedded Workbench® indicates no errors or warnings, you can continue. If errors occurred, correct them and recompile the program.

11. Initiate a debug session by clicking the **Restart Debugger** button. This downloads the compiled program into the MCU and readies the board for program execution. Click **Go** to start execution. Click **Break** (normally displayed during execution as a red upheld hand) to stop execution.



The IAR IDE screen with the Holt HI-3210 project loaded. Resized the IDE screens may be necessary.



```
/* ------------------------------------------------------------------------
 *           HOLT INTEGRATED CIRCUITS Applications Engineering
 * ------------------------------------------------------------------------
 * Copyright (c) 2011, Holt Integrated Circuits
 *
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the disclaimer below.
 *
 * Holt's name may not be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
GENERAL INFORMATION:
  The 3210 has two interrupt output pins, both are push-pull high-true outputs:
  MINT - used for auto-init error notifications,
         Receiver FIFO's interrupts,
         Transmitter Ready (TX0-TX3)
  AINT - used for interrupts in ARINC label memory. Keep in mind when using this
      interrupt the message should be read within about 340us (at fast speed) if constant
      messages back to back are expected on the same channel and label, otherwise message overflow will occur. If more
      timing flexibility is desired use the FIFO's with the MINT interrupt. Up to 32 messaged
      can be received and stored up before any messages are lost without reading them.


  The Atmel Cortex M3 runs at 96MHz on the base board (bottom).
  The SPI clock frequency controlling the 3210 is set for 16MHz.


 * ==========================================================================
 * Version 1.0 - Compiled with IAR EWARM 7.1
 * June 2016
 * HI-3210 CTX demo program

   OPTION DIP switch 6 is used to set or clear TX0_HIZ and TX1_HIZ input pins
        of U5 and U6 (HI-8596) to demonstrate how to configure the driver for Hi-Z mode.
        This is only checked in the main idle loop.

   DIP Switches HI-SP MODE3-0 are not used in this version of the demo code. They
        are available to the user.

    DIP Switches SW1 MODE2:0 are used to configure the 3210 startup according to
    the diagram on page 27 of the data sheet. The MODE0 pin, also known as the PROG pin
    is used to initiliaze the EEPROM and initialize from EEPROM afterwards from RESET.

 *
 * DISCLAIMER: THIS SOFTWARE IS PROVIDED BY HOLT "AS IS" AND ANY EXPRESSED OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
Messages
Building configuration: at91sam3u-ek - HI-3210
Updating build tree...
main.c
Linking
at91sam3u-ek.out
Converting

Total number of errors: 0
Total number of warnings: 0
```

Project File List with Selected Descriptions

HEADER FILES WITHOUT CORRESPONDING C FILES

device_3210.h

Contains all macros for the HI-3210 including define statements for registers, control register bits, Opcodes and selected table start addresses.

3210_initialization.h

Definitions for a few configuration settings.

C FILES WITH CORRESPONDING HEADER FILES

Most of the function names are self-explanatory.

main.c

The primary program entry and main loop in `main()`.

Board_3210.c

Contains macros for SPI initialization, LEDs and various I/Os. Timer1 tick 1ms initialization and interrupt handler.

Driver_2130.c (HI-3210 low level drivers)

```
unsigned char Read_FastReg(unsigned char reg, unsigned char irq_mgmt);
void Read_AllFastRegs(unsigned char *bufferRX, unsigned char irq_mgmt);
void Write_FastMCR(unsigned short data, unsigned char irq_mgmt);
void Write_MAP (unsigned short addrMAP,unsigned char irq_mgmt);
unsigned short Read_MAP (unsigned char irq_mgmt);
void Write_MAP_Word (unsigned short word,unsigned char irq_mgmt);
unsigned short Read_MAP_Word( unsigned char irq_mgmt);
void Read_FIFO(unsigned short FIFOChan, unsigned char *bufferRX,
unsigned char irq_mgmt);
void Write_MAP_Byte (unsigned char byte,unsigned char irq_mgmt);
unsigned char Read_MAP_Byte( unsigned char irq_mgmt);
unsigned char ReadByteFromMAP( unsigned short mapAddr, unsigned char
irq_mgmt);
void TransmitMessage(unsigned char chan, unsigned char *bufferTX,
unsigned char irq_mgmt);
void WriteRegister(unsigned short mapAdr, unsigned char value);
void Memory_watch(unsigned short address);
void Configure_ARM_MCU_SPI(void);
void Memory_watchRange(unsigned short address, unsigned short range);
void Read_ARINC_Block(unsigned short Chan, unsigned char *bufferRX,
unsigned char irq_mgmt);
```

```
void Read_ARINC_Mesg(unsigned short Chan, unsigned char *bufferRX,
unsigned char irq_mgmt);
```

Note: irq_mgmt is not used in this demo.

3210_Interrupts.c

Initialization and IRQ handler for message interrupt on AINT. See demo 1.

3210Demo.c (these demos are called mostly from the console)

```
void displayRegisters2(void);
void displayRXControlRegs(void);
void Read_AllFastRegs(unsigned char *bufferRX, unsigned char
irq_mgmt);
void setTX_HiZ(unsigned int ctl);
unsigned int getHi_SP_SW(void);
unsigned char getModeDIPs(void);
void setLED(unsigned int ctl, unsigned int on);
void bitSetPortB(unsigned int bit);
void bitClearPortB(unsigned int bit);
void bitSetPortC(unsigned int bit);
void bitClearPortC(unsigned int bit);
void pulseAACK(void);
void pulseMINTACK(void);
void displayRegistersSet(void);
unsigned short getConsoleAddress(void);
void TX_TransmitDirectandFIFOreads(void);
unsigned int get32Bit(void);
void ARINC_RAM_Reception(unsigned char loopback);
void ARINC_FIFO_Reception(void);
void ARINC_LoadMessage(unsigned short tx_chan, unsigned short
seq, unsigned char *action, unsigned char *value, unsigned char
mesgCnt);
void readAndDisplayFIFOs(void);
void hotKeys(unsigned char ch);
void TX0_Simple_Scheduler( unsigned char rate);
void RepeaterDemo(void);
void Concentrator_Demo(void);
void programEEPROM(void);
void setLabel(unsigned short ch, unsigned char label);
void clearLabel(unsigned short chTblOffset, unsigned char label);
void transmitTX2_TX3(void);
```

console.c

Console functions used by all demos.

```
ConfigureUsart1();
```

```
     Show_menu();
     chk_key_input(); Key entry detect and demo execution handler.
```

Application Development Kit Notes

The HI-3210 was designed for compatibility with microcontrollers or FPGAs having a Serial Peripheral Interface (SPI). Device RAM and register locations are written or read with the help of 8-bit SPI commands. Most read or write operations use a Memory Address Pointer (MAP) to designate the address of the next location accessed. To speed up multi-word transfers, the Memory Address Pointer automatically increments to the next address after each word read or write is performed. Register addresses 0x8000 – 0x800F can be read or written directly using a Fast Access op code, without using the memory address pointer.

The HI-3210 data transfer speed depends on the SPI clock frequency provided by the MCU SPI interface. The ARM Cortex M3 MCU master clock frequency is 96MHz, using an internal PLL. MCU peripherals divide this clock by two for a 48MHz clock source. The SPI block divides this by three for a 16MHz SPI clock. The HI-3210 has a maximum SPI clock frequency of 20MHz.

The demo board includes a fully featured high-end USB debugger port. No additional debug tools are required to rebuild the demo project and re-flash the ARM Cortex M3 processor.

HI-3210 use with an external MCU or FPGA

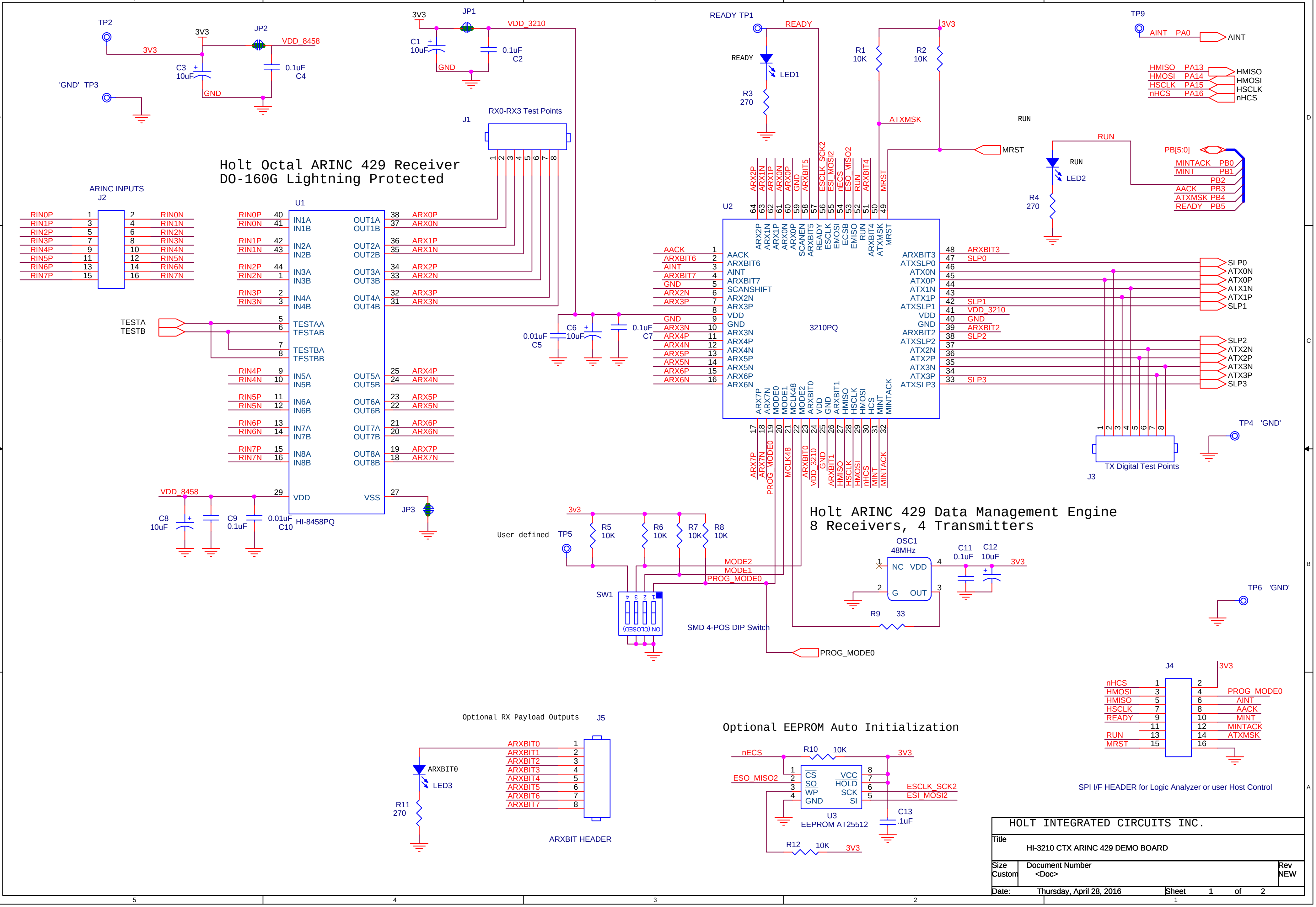
Using the HI-3210 daughter card with an external MCU or FPGA is made easy because the 48MHz oscillator is contained on this card and the essential interface signals are available on J4 header connector. See the schematic for details on those connections.

Summary

The HI-3210 is a powerful ARINC 429 multi-channel Transmitter and Receiver solution. Its high-speed 20MHz 4-wire SPI interface makes it easy to control. The HI-3210 can auto-initialize from EPPROM and operate without a FPGA or MCU in some applications. The demo program demonstrates nearly all aspects of the device and the low level C drivers make it easy to port to other platforms. A rich set of menu utilities allow easy viewing of device memory on the console, to display data accessed by the MCU through the HI-3210 SPI interface.

Bill of Materials
HI-3210 Evaluation board
Rev. A

Item	***Qty***	***Description***	***Reference***	***DigiKey***	***Mfr P/N***
1	1	PCB, Bare, Eval Board	N/A	--------	Jet Tech # 38476
2	2	Capacitor, Cer 0.01uF 20% 50V 7XR 0805	C5, C10	399-1160-1-ND	Kemet C0805C103M5RACTU
3	6	Capacitor, Cer 0.1uF 20% 50V Z5U 0805	C2,C4,C7,C9,C11,C13	399-1176-1-ND	Kemet C0805C104M5UACTU
4	8	Capacitor, 4.7uF 16V X7R 1210 Low ESR	C15,C16,C20,21,C26,C27, C28,C29	587-1392-1-ND	Taiyo Yuden EMK325B7475KN-T
5	15	Capacitor, 10uF 16V X7R 0805 Low ESR	C14,C19,C24,C25,C6,C8,C12,C17,C18,C22,C23,C30, C31,C32,C33	587-3319-1-ND	Taiyo Yuden EMK212BB7106MG-T
6	2	Capacitor 68uF 10% 6V Tant SMD 2413	C1,C3	399-3764-1-ND	Kemet T491C686K006AT
7	2	Header, Male 2x20 0.1" Pitch, 0.230" Pins, 0.120" Tails	J3,J4	S2012E-20-ND	Sullins PEC20DAAN
8	1	Header, Male 2x5 0.1" Pitch, 0.230" Pins, 0.120" Tails	J5A	S2012E-05-ND	Sullins PEC05DAAN
9	1	Header, Male 2x7 0.1" Pitch, 0.230" Pins, 0.120" Tails	J5B	S2012E-07-ND	Sullins PEC07DAAN
10	3	Header, Male 1x8 0.1" Pitch, 0.230" Pins, 0.120" Tails	J1,J5,J6	S1012E-08-ND	Sullins PEC08SAAN
11	2	Header, Male 2x8 0.1" Pitch, 0.230" Pins, 0.120" Tails	JP2, JP4	S2012E-08-ND	Sullins PEC08DAAN
12	1	IC Gate OR 4Ch 2-Inp 14-SOIC	U4	296-1235-1-ND	TI 74LVC32ADR
13	1	LED Yellow 0805	LED8(Tx)	160-1175-1-ND	Lite On LTST-C170YKT
14	6	LED Green 0805	LED1-6	160-1179-1-ND	LiteOn LTST-C170GKT
15	1	LED Red 0805	LED7(Rx)	160-1176-1-ND	LiteOn LTST-C170CKT
16	1	Osc XO 48.0MHZ HCMOS SMD	OSC1	535-10086-1-ND	Abracon ASV-48.000MHZ-E-T
17	1	Resistor, 33 5% 1/8W 0805	R9	P33ACT-ND	Panasonic ERJ-6GEYJ330V
18	8	Resistor, 270 5% 1/8W 0805	R3, R4, R11,	P270ACT-ND	Panasonic ERJ-6GEYJ271V
19	8	Resistor, 10K 5% 1/8W 0805	R1,R2,R5,R6,R7,R8,R10,R12	P10KACT-ND	Panasonic ERJ-6GEYJ103V
20	1	DIP Switch 4-Position SMD	SW1	CT2194MST-ND	CTS 219-4MST
21	1	DIP Switch 6-Position SMD	SW2	CT2196MST-ND	CTS 219-6MST
22	1	Test Point, Red Insulator, 0.062" hole	TP2 (3V3)	36-5010-ND	Keystone 5010
23	2	Test Point, Black Insulator, 0.062" hole	TP3,TP6 (GND)	36-5011-ND	Keystone 5011
24	1	Test Point, White Insulator, 0.062" hole	Ready (TP1)	36-5012-ND	Keystone 5012
25	2	Test Point, White Insulator, 0.040" hole	UserTP5), AINT (TP9)	36-5002-ND	Keystone 50012
26	1	IC EEPROM 512KBIT 20MHZ 8SOIC	U3	AT25512N-SH-B-ND	Atmel AT25512N-SH-B
27	1	IC HI-8458PQFP 44QFP	U1	Holt IC	Holt IC
28	1	HI-3210PS 64QFP	U2	Holt IC	Holt IC
29	4	HI-8597PS 16WSOIC	U5,U6,U7,U8	Holt IC	Holt IC

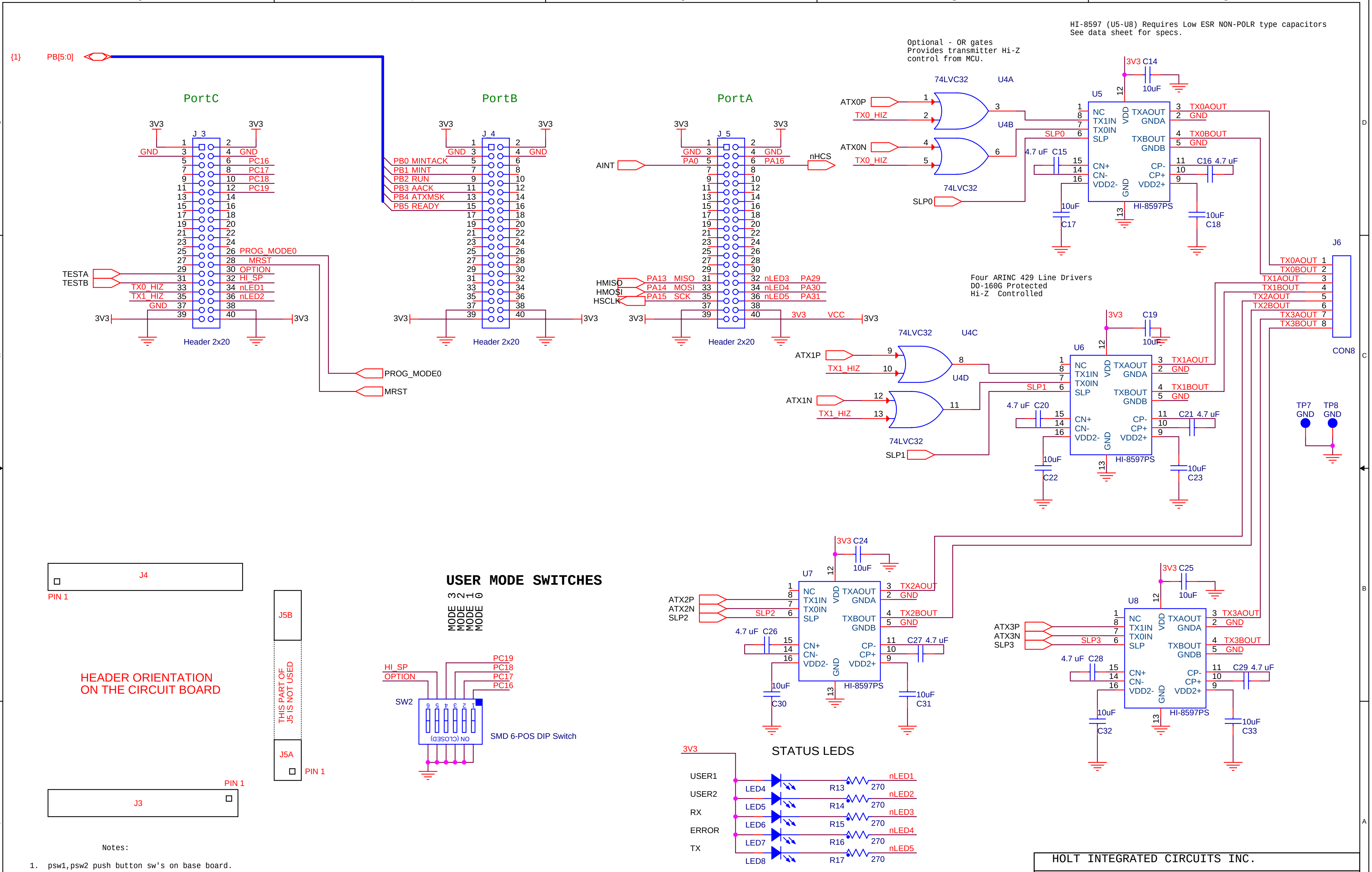


HOLT INTEGRATED CIRCUITS INC.

Title
HI-3210 CTX ARINC 429 DEMO BOARD

Size	Document Number	Rev
Custom	<Doc>	NEW

Date: Thursday, April 28, 2016 Sheet 1 of 2



Notes:

1. psw1,psw2 push button sw's on base board.

2. tx,rx,rts,cts UART signals on base board.

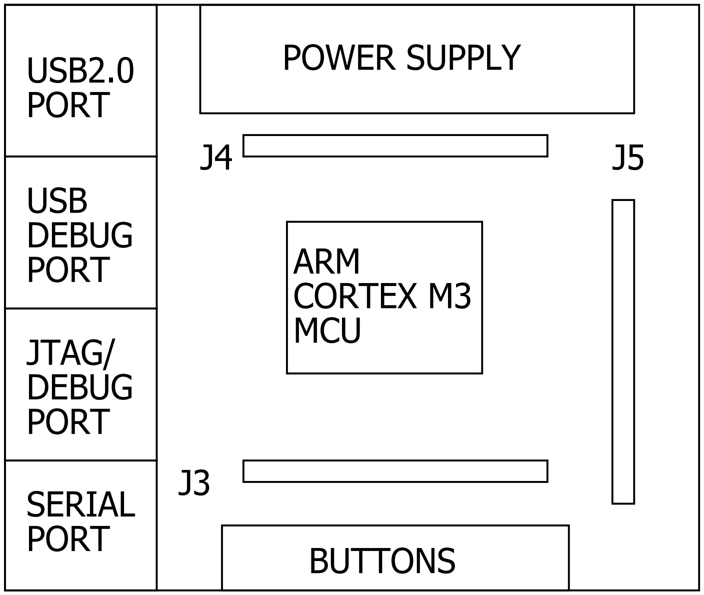
3. sw1 - sw4 spare DIP sw's on page 3.

HOLT INTEGRATED CIRCUITS INC.

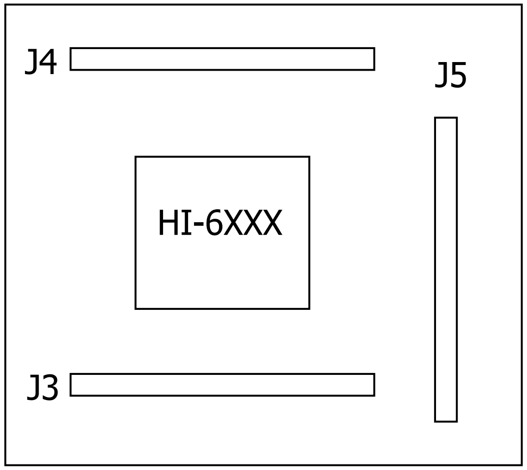
Title	HI-3210 CTX ARINC DEMO BOARD	
Size Custom	Document Number <Doc>	Rev NEW
Date: Thursday, April 28, 2016	Sheet 2 of 2	

Bill of Materials
ARM Cortex M3 MCU Board
Rev. E

Item	Qty	Description	Reference	DigiKey	Mfr P/N
1					
2	1	PCB, Bare, Evaluation Board	N/A	--------	
3	1	Ferrite Bead, 220 Ohm @ 100MHz 300mA DC 0805	FB1	732-1602-1-ND	Wurth 742792034
4	2	Capacitor, Ceramic 10nF 10% 50V X7R 0603	C1,C42	490-1512-1-ND	Murata GRM188R71H103KA01D
5	2	Capacitor, Ceramic 10pF 10% NP0 C0G 0V 0603	C23,C34	490-1403-1-ND	Murata GRM1885C1H100JA01D
6	4	Capacitor, Ceramic 20pF 5% NP0 C0G 0V 0603	C14,C21,C25, C27	490-1410-1-ND	Murata GRM1885C1H200JA01D
7	29	Capacitor, Ceramic 100nF 10% 25V Y5V 0603	C2,C4,C6-C11, C13,C15-C19,C22,C24,C26,C28,C29,C33, C35-C40,C45-46,C54	490-1575-1-ND	Murata GRM188F51E104ZA01D
8	4	Capacitor, Tantalum 4.7uF 10% 10V Low ESR SMD 1206	C5,C20,C31, C32	478-2391-1-ND	AVX TPSA475K010R1400
9	4	Capacitor, Tantalum 10uF 10% 10V Low ESR SMD 1206	C3,C12,C30,C41	478-3317-1-ND	AVX TPSA106K010R1800
10	1	Capacitor 22uF 10% 6.3V Tantalum Low ESR SMD C	C43	399-10521-1-ND	Kemet T495C226K006ATE380
11	1	Capacitor 100uF 10% 6.3V Tantalum Low ESR SMD C	C44	495-1509-1-ND	Kemet T495C107K006ZTE150
12	1	Header, Male Shrouded 2x10, 0.1" Pitch	J1	HRP20H-ND	Assmann AWHW20G-0202-T
13	1	Connector, Receptacle USB Mini B Rt-Angle PCB Mount	J2	H2959CT-ND	Hirose UX60-MB-5ST
14	1	Connector DB9F, Right-Angle PCB Short Body, Board Lock	J6	AE10924-ND	Assman A-DF-09-A/KG-T4S
15	1	Jack, DC Power, 2.5mm ID x 2.1mm pin	J7	CP-102AH-ND	Cui PJ-102AH
16	3	Receptacle, Female 2x20, 0.1" Pitch, 8.5mm Height, 3.2mm Solder Tails	J3,J4,J5	S6104-ND	Sullins PPTC202LFBN-RC
17	1	Solder Jumper	JP1	SOLDER OPEN	
18	2	Inductor, 10uH,100mA 0805	L1,L2	490-4029-1-ND	Murata LQM21FN100M70L
19	1	LED Green 0805	LED1	160-1179-1-ND	LiteOn LTST-C170GKT
20	0	Resistor, Prov 1/8W 0805	R1,R15,R16, R44,R45	DO NOT STUFF	
21	7	Resistor, 0 ohm 1/8W 0805	R9,R12,R13, R14,R22,R23, R29	P0.0ACT-ND	Panasonic ERJ-6GEY0R00V
22	2	Resistor, 1.0 5% 1/8W 0805	R7,R8	P1.0ACT-ND	Panasonic ERJ-6GEYJ1R0V
23	2	Resistor, 39 5% 1/8W 0805	R4,R5	P39ACT-ND	Panasonic ERJ-6GEYJ390V
24	1	Resistor, 150 5% 1/8W 0805	R17	P150ACT-ND	Panasonic ERJ-6GEYJ151V
25	1	Resistor, 4.7K 5% 1/8W 0805	R3	P4.7KACT-ND	Panasonic ERJ-6GEYJ472V
26	1	Resistor, 6.8K 5% 1/8W 0805	R6	P6.8KACT-ND	Panasonic ERJ-6GEYJ682V
27	0	Resistor, 47K 5% 1/8W 0805	R18	DO NOT STUFF	Panasonic ERJ-6GEYJ473V
28	0	Resistor, 68K 5% 1/8W 0805	R19	DO NOT STUFF	Panasonic ERJ-6GEYJ683V
29	11	Resistor,100K 5% 1/8W 0805	R2,R10,R11, R20,R21,R24, R25,R26,R27, R28,R42	P100KACT-ND	Panasonic ERJ-6GEYJ104V
30	3	Switch Tactile SPST 6 x 6 mm SMT	SW1,SW2,SW3	P12932SCT-ND	Panasonic EVQ-Q2B03W
31	2	Test Point, Black Insulator, 0.062" hole	TP2,TP3	5011K-ND	Keystone 5011
32	1	Test Point, Red Insulator, 0.062" hole	TP1	5010K-ND	Keystone 5010
33	1	IC, MCU 32-Bit 256KB Flash, 144-LQFP	U1	ATSAM3U4EA-AU-ND	Atmel ATSAM3U4EA-AU
34	1	4-Ch TVS ESD Protection SOT23-6	U2	296-28203-1-ND	TI TPD4E001DBVR
35	1	IC, RS232 Driver/Receiver 3.0 to 5.5VDC 16-SOIC (3.9mm wide)	U3	296-19752-1-ND	Texas Inst MAX3232EIDR
36	1	IC Voltage Regulator 3.3V 1A LDO, SOT-223	U5	497-1228-1-ND	ST Micro LD1117AS33TR
37	1	PolyZen 5.6V PPTC protected Zener SMD	U6	ZEN056V130A24LSCT-ND	TE ZEN056V130A24LS
38	1	Filter, EMI 35dB 10A 1MHz-1GHz SMD	U7	490-5052-1-ND	Murata BNX022-01L
39	1	IC Voltage Ref 2.5V 1% Micropower SOT-23	VR1	576-1047-1-ND	Micrel LM4040DYM3-2.5
40	1	Crystal 12.00MHz, 50ppm 20pF, HC-49US leaded	Y1	631-1105-ND	Fox FOXSLF/120-20
41	1	Crystal, 32768 Hz 12.5pF cylinder leaded	Y2	535-9033-1-ND	Abracon AB26TRB-32.768KHZ-T
42	5	Rubber Foot, Bumpon Black Hemisphere, .312 X.200 H	Place at 4 corners and center	SJ5746-0-ND	3M SJ61A1
47	1	Capacitor, Ceramic 100nF, -20% / +80% 25V Y5V 0603	C66	490-1575-1-ND	Murata GRM188F51E104ZA01D
48	1	Capacitor, Ceramic 33pF, 5% 50V C0G 0603	C59	490-1415-1-ND	Murata GRM1885C1H330JA01D
49	2	Capacitor, Ceramic 15pF, 5% 50V C0G 0603	C60,C61	490-1407-1-ND	Murata GRM1885C1H150JA01D
54	1	Ferrite Bead, 220 Ohm @ 100MHz 300mA DC 0805	FB2	732-1602-1-ND	Wurth 742792034
55	1	Solder Jumper	JP2	SOLDER OPEN	
56	1	Connector, Receptacle USB Mini B Rt-Angle PCB Mount	J8	H2959CT-ND	Hirose UX60-MB-5ST
57	1	LED Green 0805	LED2	160-1179-1-ND	LiteOn LTST-C170GKT
59	1	Resistor, 220 ohm 5% 1/10W 0603	R31	P220GCT-ND	Panasonic ERJ-3GEYJ221V
63	2	Resistor, 27 ohm 5% 1/10W 0603	R36,R38	P27GCT-ND	Panasonic ERJ-3GEYJ270V
66	1	4-Ch TVS ESD Protection SOT23-6	U4	296-28203-1-ND	TI TPD4E001DBVR



LOWER CIRCUIT BOARD



STACKING UPPER CIRCUIT BOARD

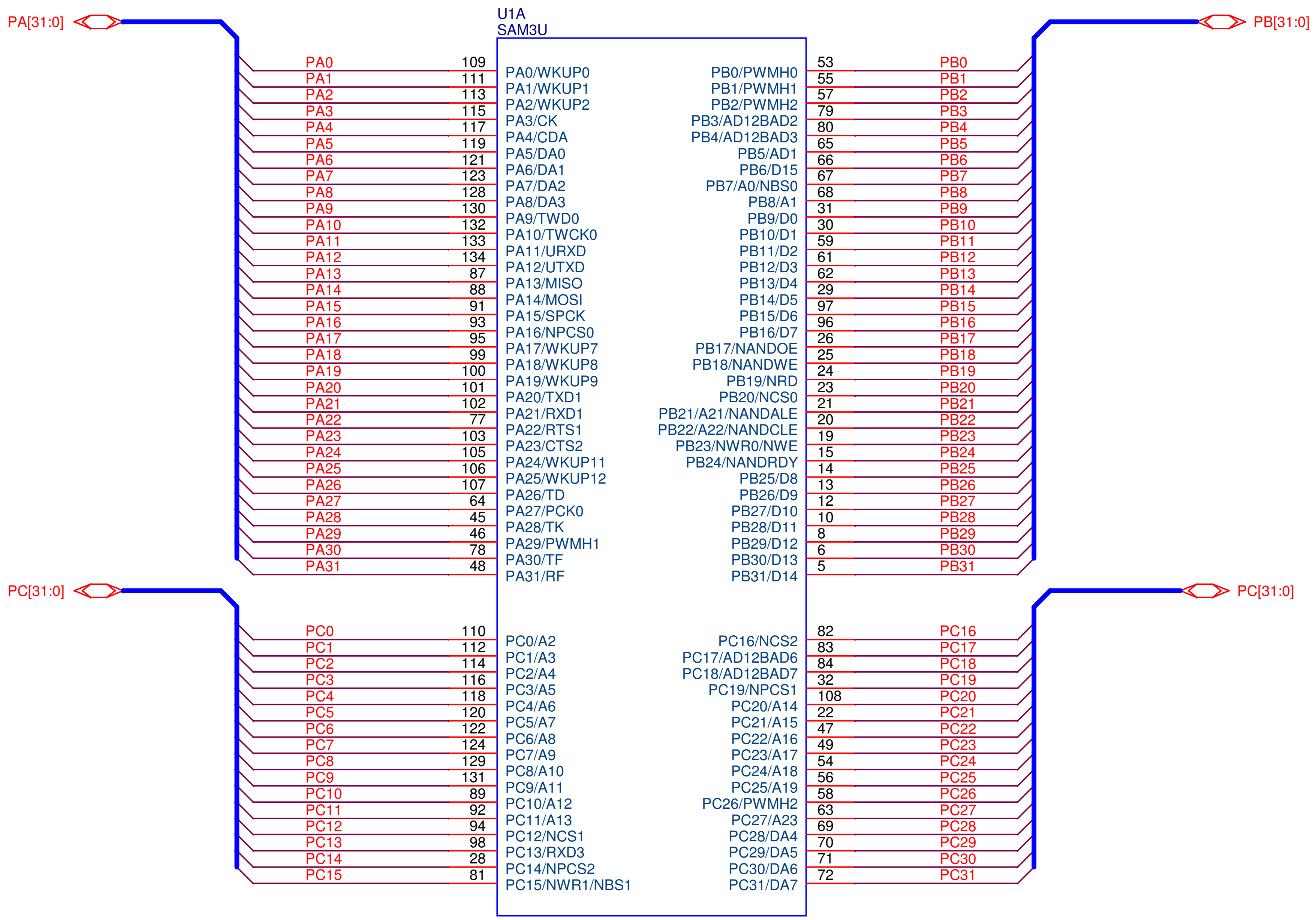
J3,J4 & J5 ARE DUAL-ROW STACKING RECEPTACLES (LOWER BOARD) AND HEADERS (UPPER BOARD).

HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title
ARM CORTEX M3 MICROCONTROLLER BOARD

Size	Document Number	Rev
A	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 1 of 7



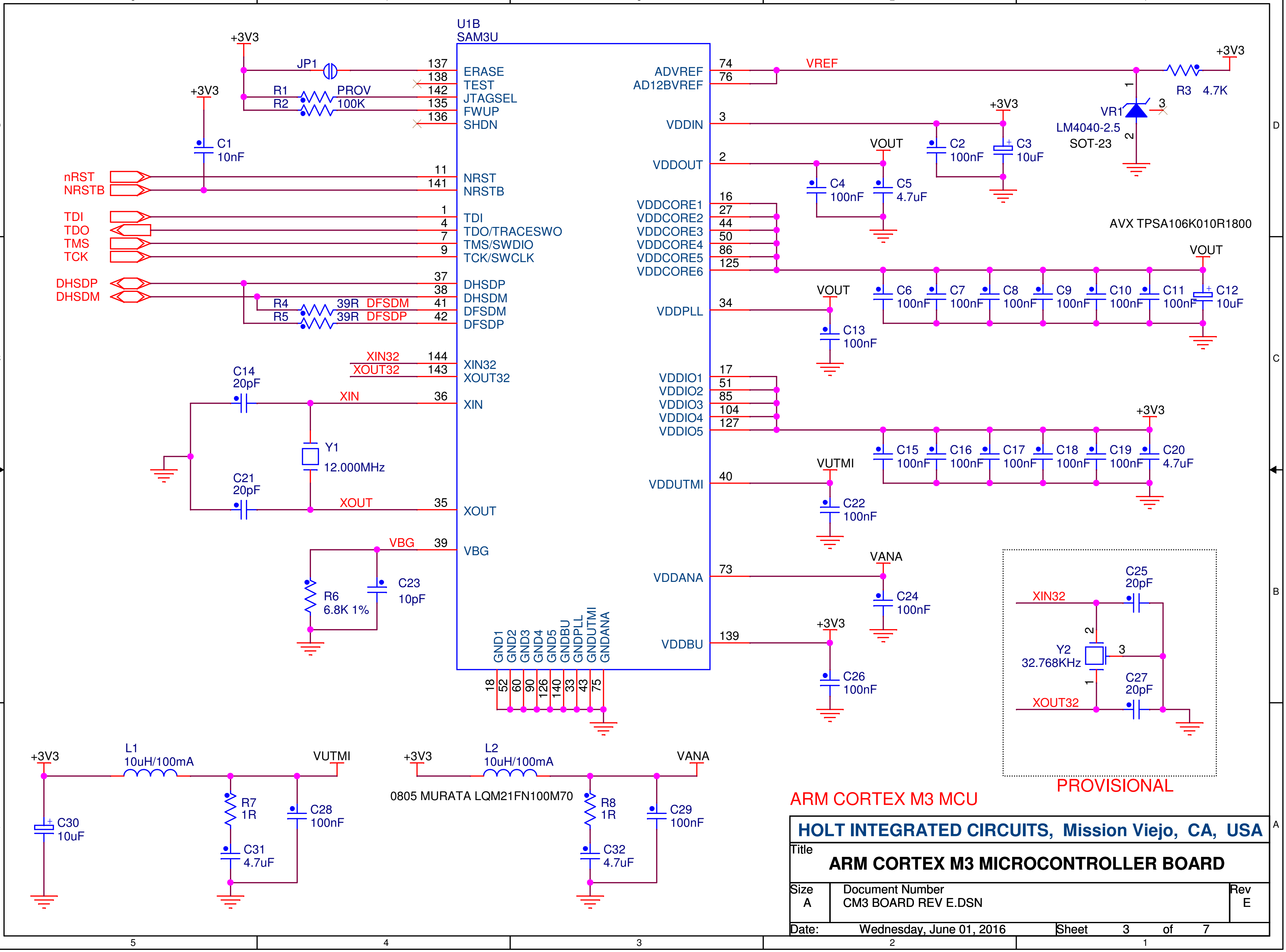
ARM CORTEX M3 PIO

HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title
ARM CORTEX M3 MICROCONTROLLER BOARD

Size	Document Number	Rev
A	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 2 of 7



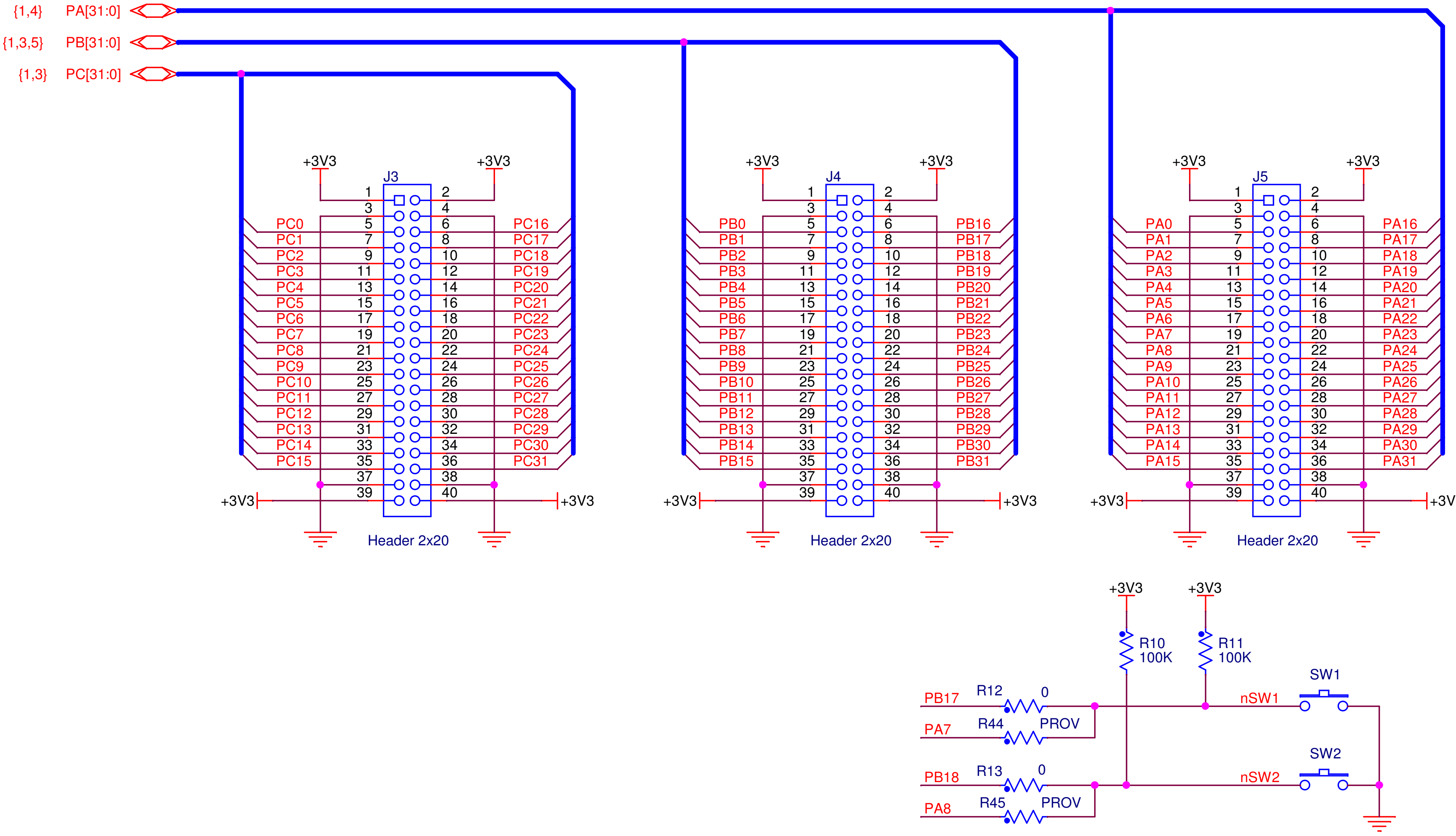
HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title

ARM CORTEX M3 MICROCONTROLLER BOARD

Size	Document Number	Rev
A	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 3 of 7



BOARD I/O HEADERS, BUTTONS

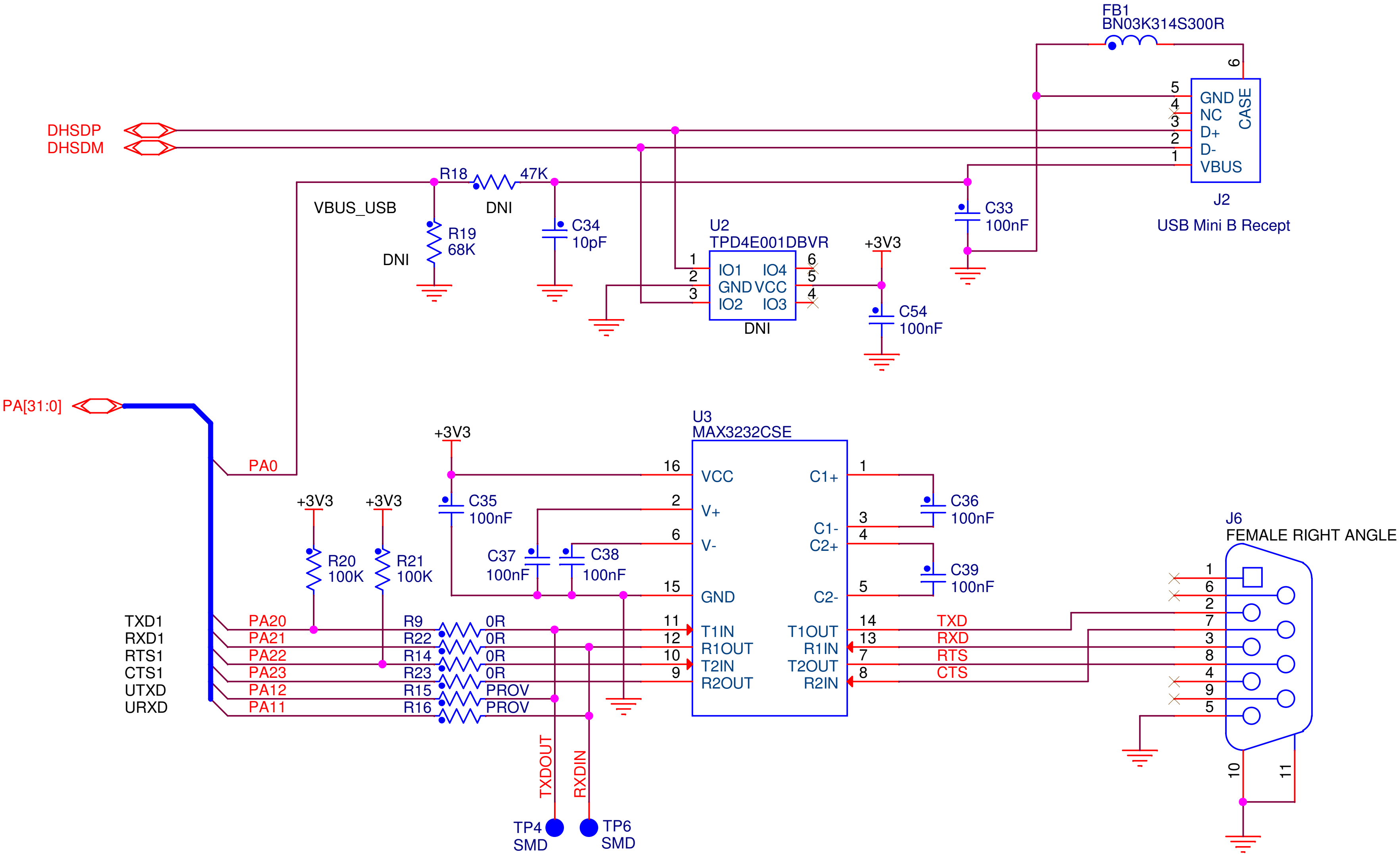
HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title

ARM CORTEX M3 MICROCONTROLLER BOARD

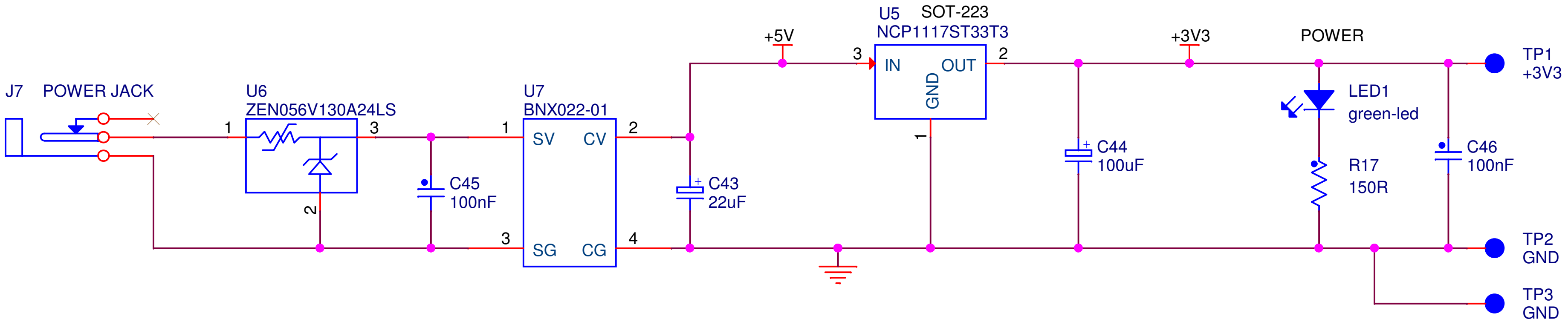
Size	Document Number	Rev
A	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 4 of 7



USB & RS-232 SERIAL

HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA		
Title **ARM CORTEX M3 MICROCONTROLLER BOARD**		
Size A	Document Number CM3 BOARD REV E.DSN	Rev E
Date: Wednesday, June 01, 2016	Sheet 5 of 7	



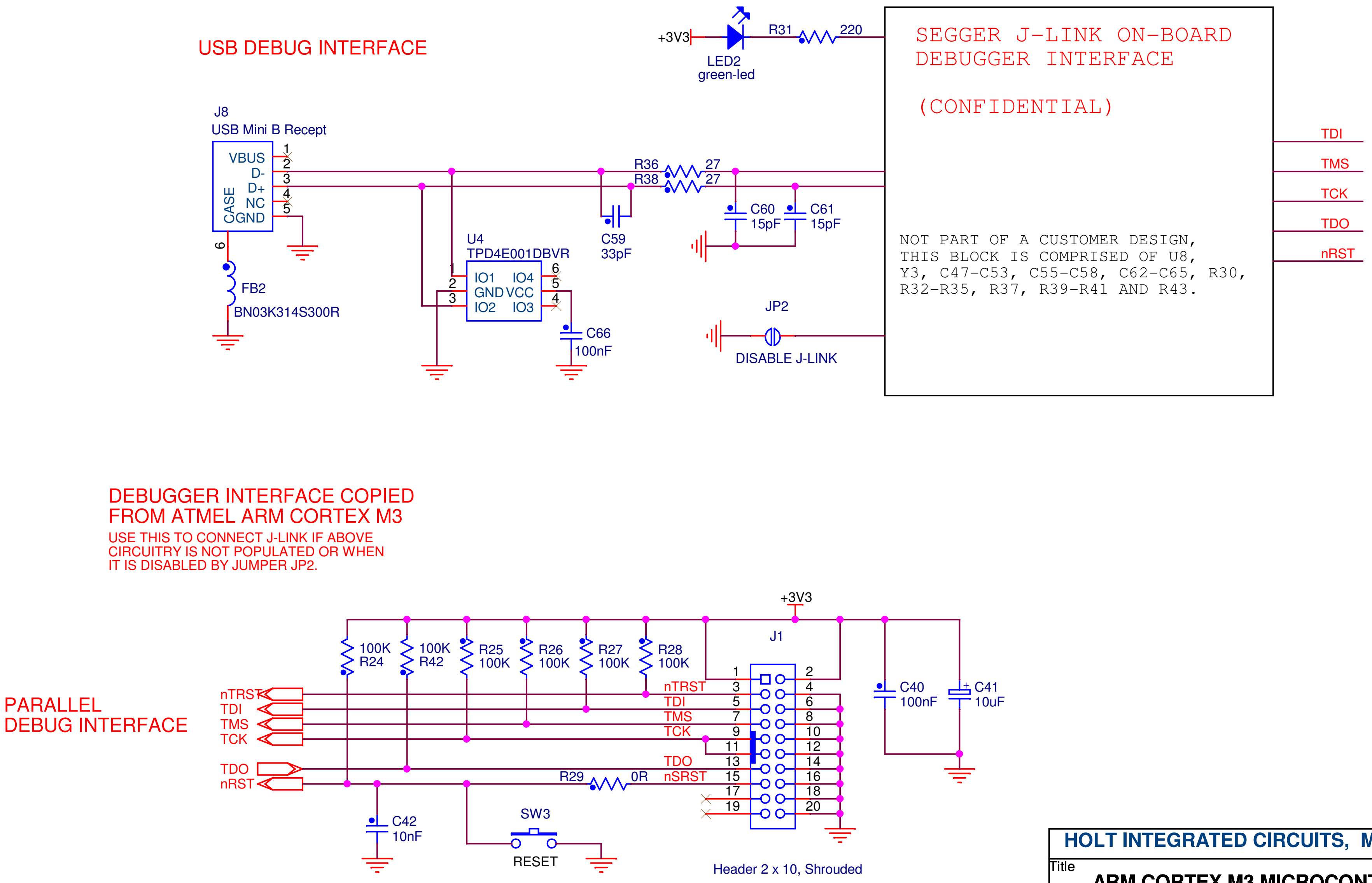
POWER SUPPLY

HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title
ARM CORTEX M3 MICROCONTROLLER BOARD

Size	Document Number	Rev
A	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 6 of 7



SEGGER J-LINK ON-BOARD DEBUGGER INTERFACE

(CONFIDENTIAL)

NOT PART OF A CUSTOMER DESIGN, THIS BLOCK IS COMPRISED OF U8, Y3, C47-C53, C55-C58, C62-C65, R30, R32-R35, R37, R39-R41 AND R43.

DEBUGGER INTERFACE COPIED FROM ATMEL ARM CORTEX M3

USE THIS TO CONNECT J-LINK IF ABOVE CIRCUITRY IS NOT POPULATED OR WHEN IT IS DISABLED BY JUMPER JP2.

HOLT INTEGRATED CIRCUITS, Mission Viejo, CA, USA

Title

ARM CORTEX M3 MICROCONTROLLER BOARD

Size	Document Number	Rev
Custom	CM3 BOARD REV E.DSN	E

Date: Wednesday, June 01, 2016 Sheet 7 of 7