SAND2023-00068

SANDIA REPORT
SAND20XX-XXXX
Printed Click to enter a date

Sandia National Laboratories

High Temperature Component and Data Link Evaluation

Andrew A. Wright, Avery T. Cashion, and Francis Tiong

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

ABSTRACT

Characterizing and monitoring the long term performance of a geothermal well to determine it energy production can be challenging because downhole equipment and the well itself will degrade over time. Subsurface tools are lowered down the well to measure temperature, pressure, pH, and other parameters, but high temperatures, high pressures, and great depths pose many challenges to these instruments. Sensors used for measurements generate a small voltage signal and cannot relay the data across the great depths of the well. Thus, local data processing is required to convert the low analog signal to a digital signal that can be transmitted over greater lengths. However, there are very few components on the market that can survive such a harsh environment. In this study, we evaluated multiple high temperature microcontrollers capable of operating above 210°C. We examined five microcontrollers (HT83C51, SM320F2812-HT, SM470R1B1M-HT, SM320F28335-HT, and RC10001), documented the results, and provided recommendations regarding which should be used in today's geothermal tools. In addition, one microcontroller was used to construct a high temperature data link for communicating across a single conductor wireline. Due to the limited memory of the RC10001, the RC2110836 memory chip was also evaluated. We recommend the SM470R1B1M-HT, SM320F28335-HT, and RC10001 with RC2110836 microcontrollers for geothermal applications. The SM320F28335-HT was used to demonstrate a high temperature data link unit. That data link has shown 30 kbps data rates across 1524 m (5000 ft) of single conductor wireline up to 170°C.

ACKNOWLEDGEMENTS

This research was funded by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Geothermal Technologies Program. The authors wish to acknowledge the support of our colleagues and reviewers in the preparation of this report including Giorgia Bettin, Manuel Montano, Alfred Cochrane, Lori Dotson, Johnny Silva, and Jason Krein.

EXECUTIVE SUMMARY

Subsurface logging tools that measure temperature, pressure, pH, resistance, chemistry, radiation, and other parameters are critical for characterizing a geothermal well or for monitoring long-term well performance. However, capturing these measurements can be challenging due to the high temperatures, extreme depths, high pressures, and corrosive environments associated with a geothermal field. In addition, these sensors generate a small voltage signal that cannot be detected at the surface due to the long lengths of cable it must travel. These logging tools must be capable of operating at depths greater than 610 m (2000 ft), requiring a local processor to capture the signal directly. The signal data is then either saved locally or transmitted back to the ground surface, which requires a microcontroller. Sandia National Laboratories (SNL) evaluated five commercially available microcontroller units (MCUs) capable of withstanding temperatures in excess of 210°C and used one in the development of a high temperature (HT) data link. This report will review the steps taken to evaluate these MCUs and design the data link and then present our results. This technology can enable greater sensor counts, decrease costs for real-time sensor data, and improve data accuracy.

Five HT MCUs (HT83C51, SM320F2812-HT, SM470R1B1M-HT, SM320F28335-HT, and RC10001) were evaluated. The RC2110836 memory chip was also used due to the limited memory of the RC10001. Based on our results, we recommend the SM470R1B1M-HT, SM320F28335-HT, and RC10001 with RC2110836 for geothermal applications. The SM320F28335-HT was also used to demonstrate an HT data link unit, which has produced 30 kbps data rates across a 1524 m (5000 ft) of single conductor wireline up to 170°C. These five MCUs were chosen for evaluation because it is the very few options on the market that can operate above 210°C.

SNL designed and evaluated a data link with the SM320F28335-HT microcontroller. It remained operational throughout a 1524 m (5000 ft) single conductor wireline for up to 170°C. It could not be operated up to 210°C due to limitations with the amplifier stage. However, removing the amplifier and line driver stages from the data link and directly connecting the oscilloscope to the output of the digital-to-analog converter (DAC) enabled transmission of messages in temperatures up to 210°C. Using the orthogonal frequency division multiplexer (OFDM) and binary phase shift keying (BPSK) communication technique, 30 kbps data rates were demonstrated at elevated temperatures. The printed circuit board (PCB) was designed to accommodate voltage regulation and access pins for the analog-to-digital converter (ADC) to allow power transfer through the wireline and to enable data to be received in future testing.

One of the major problems was package failures, which severely limited the testing on the RC10001 microcontroller. We observed deformation of the PCB, degradation of the solder, mismatch of the coefficient of thermal expansion coefficient of thermal expansion (CTE), delamination of the integrated circuits (ICs), and shorting of internal traces. The reliability of the packaging will be improved by replacing the Rogers PCB with a ceramic PCB, gold/tin solder bonding or wire bonding, and an HT conformal coating. Ceramic can withstand temperatures greater than 300°C, making it prone to degradation in these applications. Gold/tin solder would be ideal for the gold pins on the IC because the solder joint will not degrade. The alternative is wire bonding, which fuses gold wire to the pins and to the PCB pads. The wire is capable of operating at temperatures above 300°C and allows mechanical movement between the IC and PCB (CTE mismatch) at high temperatures. Finally, applying a conformal coating, consisting of a layer of an applied cured liquid epoxy covering the IC and PCB, can improve the strength between the IC and PCB as well prevent

oxidation to the metals. These packaging improvements can significantly improve the reliability of the logging tool electronics that will be exposed to high temperatures, mechanical shock, and vibration.

Future improvements to the data link can be conducted to improve data rates, operating temperatures, signal-to-noise ratio (SNR), error rates, and receiving data. The required operating temperatures can be achieved by replacing the SM320F28335-HT with the RC10001. Using an ADC enables signals to be received from the surface. Data rates can be significantly increased by increasing the constellation point by incorporating quadrature amplitude modulation (QAM) instead of BPSK. However, the signal integrity will need to be improved to use QAM. Signal integrity can be improved by channel coding, error correction coding, constant signal propagation profiling, increasing amplifier gain, and using pulse width modulation (PWM) DAC. Channel coding would include a few coding steps that work together to mitigate the effects of channel distortions. These steps may include interleaving the data followed by applying error correction code. Error correction techniques, such as Bose-Chaudhuri-Hocquenghem, can be incorporated into the code to improve the SNR of the system. The profile of wireline drops above 13 kHz can be compensated with the amplifier. Using a constant propagation profile, the amplitude modulation will be constant to the maximum desired operating bandwidth. Increasing the gain performance of the amplifier can further push the signal above the noise floor, mitigating noise degradation. Final methods to improve signal integrity could be replacing the resistor array DAC with a PWM DAC. This method would not experience the inaccuracies of each resistor because the resistors are not dividing the rail voltage accurately and evenly, which contribute to some noise being injected into the signal.

These improvements to the data link will allow reliable real-time data collection up to 300˚C. This technology can be used in various applications such as subsurface array systems capturing tracer data or acoustics.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND TERMS

Acronym/Term	Definition
ADC	analog-to-digital converter
ASIC	application specific integrated circuit
BPSK	binary phase shift keying
CAD	computer-aided design
CPU	central processing unit
CTE	coefficient of thermal expansion
DAQ	data acquisition
DSP	digital signal processor
F28335	SM320F28335-HT
FPGA	field programmable gate array
HT	high temperature
I/O	input/output
IC	integrated circuit
JTAG	Joint Test Action Group
MCU	microcontroller unit
N/A	not applicable
NI	National Instruments
NMOSFET	N-channel metal-oxide-semiconductor field-effect transistors
OFDM	orthogonal frequency division multiplexer
opamp	operational-amplifier
PCB	printed circuit board
PWM	pulse width modulation
QAM	quadrature amplitude modulation
RAM	random-access memory
RC	RelChip
SMD	surface mount device
SNL	Sandia National Laboratories
SRAM	static random-access memory
T_d	decomposition transition
T_g	glass transition
TI	Texas Instruments

1. INTRODUCTION

Subsurface electronic tools are critical for characterizing geothermal wells and for monitoring their long-term performance. The high temperatures, extreme depths, high pressures, and corrosive environments associated with a geothermal field make it challenging to measure various parameters, such as temperature, pressure, pH, resistance, tracers, and radiation. Logging tools and sensors temporarily used to capture these measurements generate a small voltage signal that cannot travel long distances without falling below the noise floor. Logging tools are expected to operate at depths exceeding 610 m (2000 ft), so a local processor—in this case a microcontroller—is necessary to capture the signal directly, after which the data is either saved locally or transmitted back to the surface. Due to the elevated temperatures, the microcontroller also needs to operate above 210˚C. Sandia National Laboratories (SNL), evaluated microcontrollers on the market that can operate at or beyond 210˚C in addition to developing a high temperature data link—an electronic unit used to transmit data across a data line.

Five high temperature (HT) microcontroller units (MCU) were evaluated from the manufacturers Honeywell (HW), Texas Instruments (TI), and RelChip (RC). Honeywell's HT83C51 served as a reference due to the product's reputation and SNL's prior experience using the MCU for geothermal applications. This research also evaluated TI's SM320F2812-HT, SM470R1B1M-HT, and SM320F28335-HT as well as RC's RC10001 and its associated RC2110836 memory chip. HT Data Link evaluated TI's SM320F28335-HT MCU as the processor for developing a high temperature data link. This paper includes steps to evaluating these devices, designing a data link, and the elevated temperature results observed. This technology can enable greater sensor counts, improve data accuracy, and decrease costs associated with real-time sensor data.

2. HT COMPONENTS AND HT DATA LINK

2.1. Evaluation of High Temperature Microcontrollers and Memory

SNL evaluated five commercially available microcontrollers with operating temperatures listed at 210˚C or higher as shown in Table 1. RelChip's RC10001 has comparatively limited memory for utilizing the integrated circuit (IC) in logging tools; therefore, the RC2110836 static random-access memory (SRAM) chip was also evaluated under this effort. As of June 2022, all listed MCUs are still available for purchase.

Table 1. Five HT MCUs with Performance Details

Device	Company	CPU	Max Temp. (˚C)	Max Clock (MHz)	Internal Memory	ADC	Pack-age
SM320F2812-HT	Texas Instruments ©	32-bit C2000	220	150	128Kx16 Flash, 128Kx16 ROM	16-Channel 12-bit	SMD
SM470R1B1M-HT	Texas Instruments ©	32-bit ARM7	220	60	1MB Flash, 64KB SRAM	12-Channel 10-bit	SMD
SM320F28335-HT	Texas Instruments ©	32-bit C2000	210	150 (125˚C) 100 (210˚C)	256KBx16 Flash, 34KBx16 SRAM	16-Channel 12-bit	Thru
RC10001	RelChip®	32-bit Cortex-MO	300	4	4KB SRAM	N/A	SMD
HT83C51	Honeywell©	8-bit 8051	225 (300 for 1 year)	16	8KB ROM	N/A	Thru

CPU=central processing unit
ADC=analog-to-digital converter
SMD=surface mount device
N/A=not applicable

Table 2 lists microcontrollers that can operate at 200˚C or higher, but these were not evaluated because our research focused on available components with modern microcontroller architectures capable of functioning above 210˚C. For example, TK8X51S would have been a desirable IC to evaluate; however, it is currently in the design phase and has not been fabricated at this time.

Table 2. Additional HT microcontrollers Commercially Available

Device	Company	CPU	Max Temp. (˚C)	Max Clock (MHz)	Internal Memory	ADC	Package
TK89H51B	Tekmos	8-bit 8051	210	16	1024 Byte RAM, 2K EEPROM	8-channel, 8-bit	Thru or SMD
TK8X51S	Tekmos	N/A	250	N/A	N/A	N/A	N/A
VA41600	Vorago	32-bit Cortex-M4	200	100	64KB Data, 256KB Program	8-Channel, 12-bit	SMD
VA10800	Vorago	32-bit Cortex-M0	200	50	32KB Data, 128KB Program	N/A	SMD

RAM=random-access memory

2.2. Development and Evaluation of a High Temperature Data Link

We also developed and tested a data link—an electronic system designed to receive and/or transmit data to another telecommunication unit—at elevated temperatures. Within the scope of this experiment, the data link was designed to only transmit data across a single conductor wireline, a type of high strength coaxial cable. Data links often consist of a digital signal processor (DSP), a digital-to-analog converter (DAC), and an amplifier; however, an MCU can be used in place of a DSP. This experiment used the SM320F28335-HT as the MCU for the data link as discussed below.

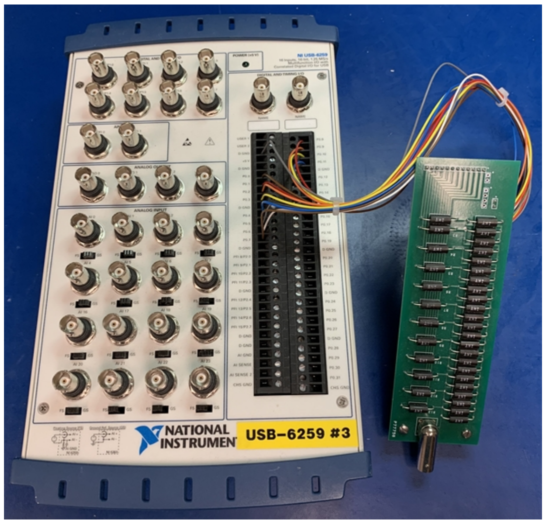


Figure 1. National Instruments DAQ with HT resistor DAC array for developing a high-speed data link.

The HT data link effort built upon a previous work conducted within the department. Under that effort, lower temperature components were used to demonstrate a high-speed data link [15]. The data link used a National Instruments data acquisition (DAQ) and a computer. Using MATLAB software, the team developed communication software that can produce an orthogonal frequency division multiplexer (OFDM) with quadrature amplitude modulation (QAM). This is a common technique used by Comcast, cell phones, Wi-Fi, and other networking that require high speed data rates. To build from the DAQ, the team developed a DAC using HT resistors, and an amplifier using Honeywell's HT transistors. The team demonstrated 3.8 Mbps across 1524 m (5000 ft) of single conductor wireline. These data rates were not observed with the latest geothermal logging tool technology. The next phase of this effort was to replace the DAQ and computer with an HT microcontroller.

2.3. Steps to Evaluating the Microcontrollers and Memory

After selecting the appropriate microcontroller, several steps were taken before testing began in an HT oven. First, we selected printed circuit board (PCB) material fit for evaluation and created a schematic based off the data sheets and/or referenced from a sister microcontroller evaluation board. Next, the PCB was designed using computer-aided design (CAD) software such as Altium. The design was fabricated and the PCB was evaluated in the oven at temperatures between 200 and 300˚C. The microcontroller was then soldered to the PCB after which it was programmed to repeat a task. Finally, after setting up the oven/DAQ, the packaged microcontroller was evaluated in the oven at 200–300˚C.

2.3.1. Selecting the Printed Circuit Board

Under this effort, the PCB must operate between 200–300˚C and was therefore evaluated for the following criteria: high glass transition temperature, high decomposition temperature, and availability for fabrication from standard PCB fabrication companies. This device was solely tested in an oven and not be exposed to vibration, greatly simplifying the packaging design. Several materials in Table 3 were considered. Two materials were ultimately selected for packaging. Rogers 3003 was used on the SRAM chip and Rogers 4003 for the microcontrollers. These were selected because they were readily available from Advanced Circuits' PCB foundry.

Table 3. Standard PCBs Available by Advanced Circuits

Material	Layer Count	Company	T_g (˚C)	T_d (°C)	Dielectric Constant
Rogers 3003	20	Rogers Corporation©		500	3
Rogers 3035	20	Rogers Corporation©		500	3.5
Rogers 3006	20	Rogers Corporation©		500	6.15
Rogers 3010	20	Rogers Corporation©		500	10.2
Rogers 4003C	20	Rogers Corporation©	280	425	3.55

Material	Layer Count	Company	T_g (˚C)	T_d (°C)	Dielectric Constant
Rogers 5870	8	Rogers Corporation©		500	2.33
Rogers 5880	8	Rogers Corporation©		500	2.2
NF-30		Taconic®		515	3
TLX-8		Taconic®		535	
Rogers Cuclad 250	20	Rogers Corporation©		500	2.97
Rogers CTLE	20	Rogers Corporation©		487	3

T_g=decomposition transition
T_d=glass transition

2.3.2. *Designing Circuit Board*

After selecting the PCB, this effort created circuits around each device based on the device's datasheet and evaluation board. The RC10001 was the only device whose evaluation board had a publicly available schematic. The listed TI MCUs did not have evaluation boards; however, their lower temperature sister chips did. As such the low temperature evaluation boards were used as a reference for the HT PCBs. The design consisted of the microcontroller and minimal external components, with most microcontrollers requiring only a small number of resistors.

2.3.2.1. HT83C51 Circuit

SNL has utilized the HT83C51 microcontroller in logging tools for over a decade. Although the chip was primarily used for 225°C applications previously, this experiment used it as a reference against newer microcontrollers, testing it at temperatures up to 300°C. SNL developed a custom evaluation board for the microcontroller to program the device. The evaluation board consists of a custom FPGA to interface with the microcontroller, multiple memory chips, and power regulation. Considering its availability, a simple evaluation board was needed, so that only the chip could be placed in the oven. The breakout board circuit is shown in Figure 2. It consists of the MCU, two power line filter capacitors, and thru hole pins for the HT wires that were routed outside the oven to the evaluation board.

2.3.2.2. RC10001 Circuit

Originally released in April 2018, the RC10001 is one of two MCUs that can operate at 300°C—the other being the HT83C51, which uses older architecture with minimal features. RelChip supplies a low temperature evaluation board specifically for the RC10001 microcontroller combined with the RC2110836 SRAM. This circuit was simplified to just the RC10001 and only the necessary four resistors. Figure 3 shows a schematic for the microcontroller. For programming purposes, a Joint Test Action Group (JTAG) for programming the device was utilized. The board also contains power line filter capacitors and thru hole pins for the HT wires.

2.3.2.3. RC2110836 Circuit

Initially, a custom evaluation board was designed for the RC2110836 SRAM IC, which was a single board with the memory chip and a microcontroller. The NUC100VD3AN microcontroller was used to interface between the SRAM and a computer. Figure 6 shows a schematic for the microcontroller. The board was designed to be about two feet in length to allow the microcontroller to be placed outside the oven while the memory is placed within. This simplified the routing of the 68 pins of the memory chip. The issue with this approach is the long traces delaminated from the board as the temperature reached above 200˚C, which caused open or short circuits. To resolve the issue, a breakout board was made solely for the memory chip, which then connected to the evaluation board via HT wire. The circuit of the breakout board is shown in Figure 7. The breakout board routed the IC pins to header pins on the outer edge. The board also contained power line filter capacitors.

2.3.2.4. SM320F2812-HT Circuit

Originally released around June 2009, the SM320F2812-HT is a highly desirable chip for its built-in DSP, which can be used to efficiently communicate between the logging tool and the surface computer. A potential drawback is size, the IC is the largest among the microcontrollers being evaluated. Although this board has a sister microcontroller, TMS320F2812, that board was designed over a decade ago and interfaces with a computer via a parallel port. Parallel ports were commonly used before the USB protocol. This board was also designed to be programmed with an external programmer via a JTAG. The evaluation board was simplified to basic components, just the microcontroller with optional capacitors. Necessary resistors were then added to the thru hole pins along the edge of the board. The resistors were not hard wired to the board because of programming issues reviewed in a later section.

2.3.2.5. SM470R1B1M-HT Circuit

The SM470R1B1M-HT is desirable for geothermal logging tools because it is the smallest >220˚C microcontroller on the market. SM470R1B1M-HT's circuit was designed around its datasheet, which was originally released in September 2009, and the TI TMS470R1B512 evaluation board. TMS470R1B512 is a sister chip with the same MCU architecture and additional features. The evaluation board used an old programming connector protocol but can also be programmed via JTAG. A schematic of the HT board is shown in Figure 5. It has multiple resistors to set the microcontroller, two JTAG options, jumper pins to change settings, and several filtering capacitors.

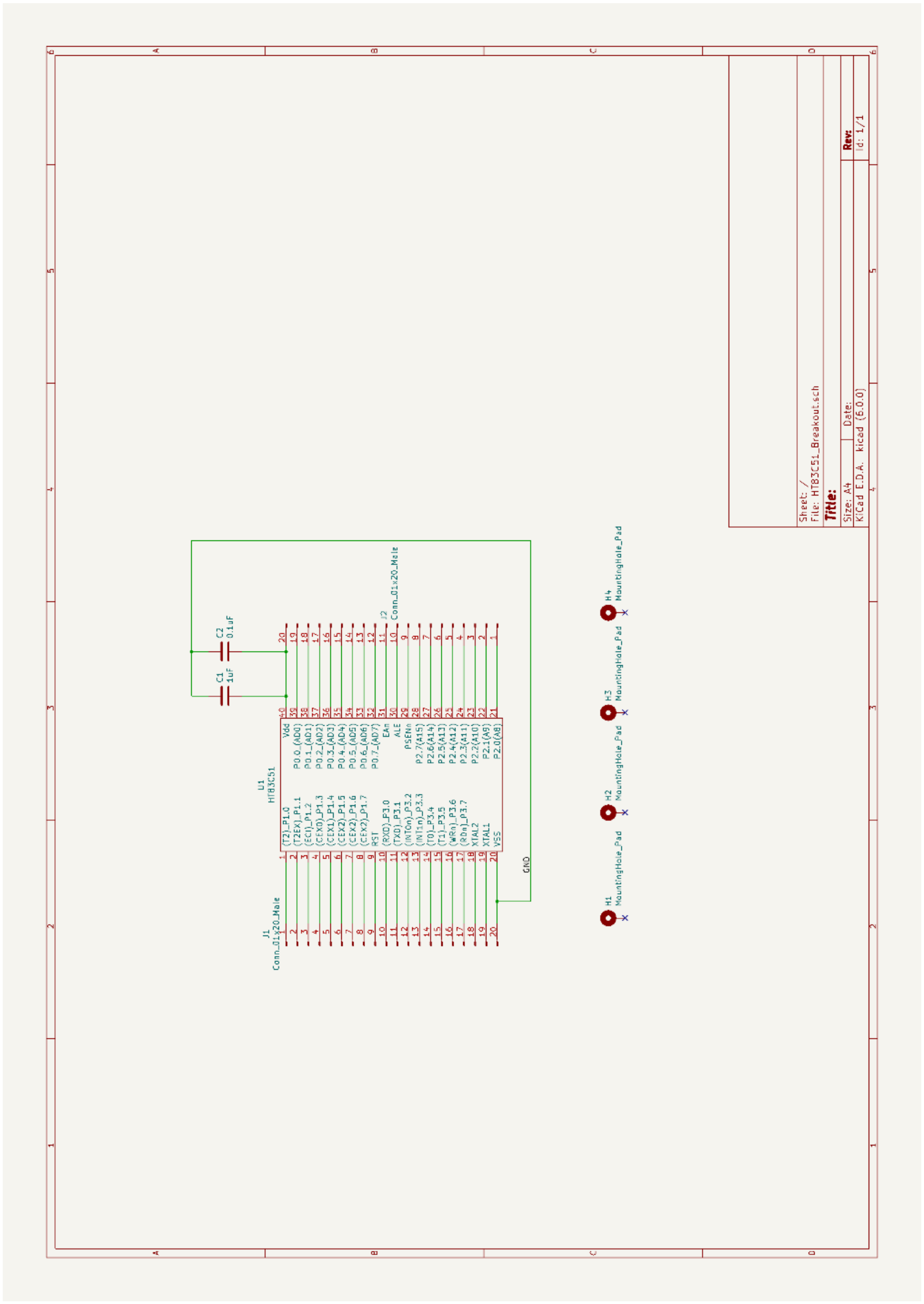


Figure 2. HT83C51 circuit for breakout board.

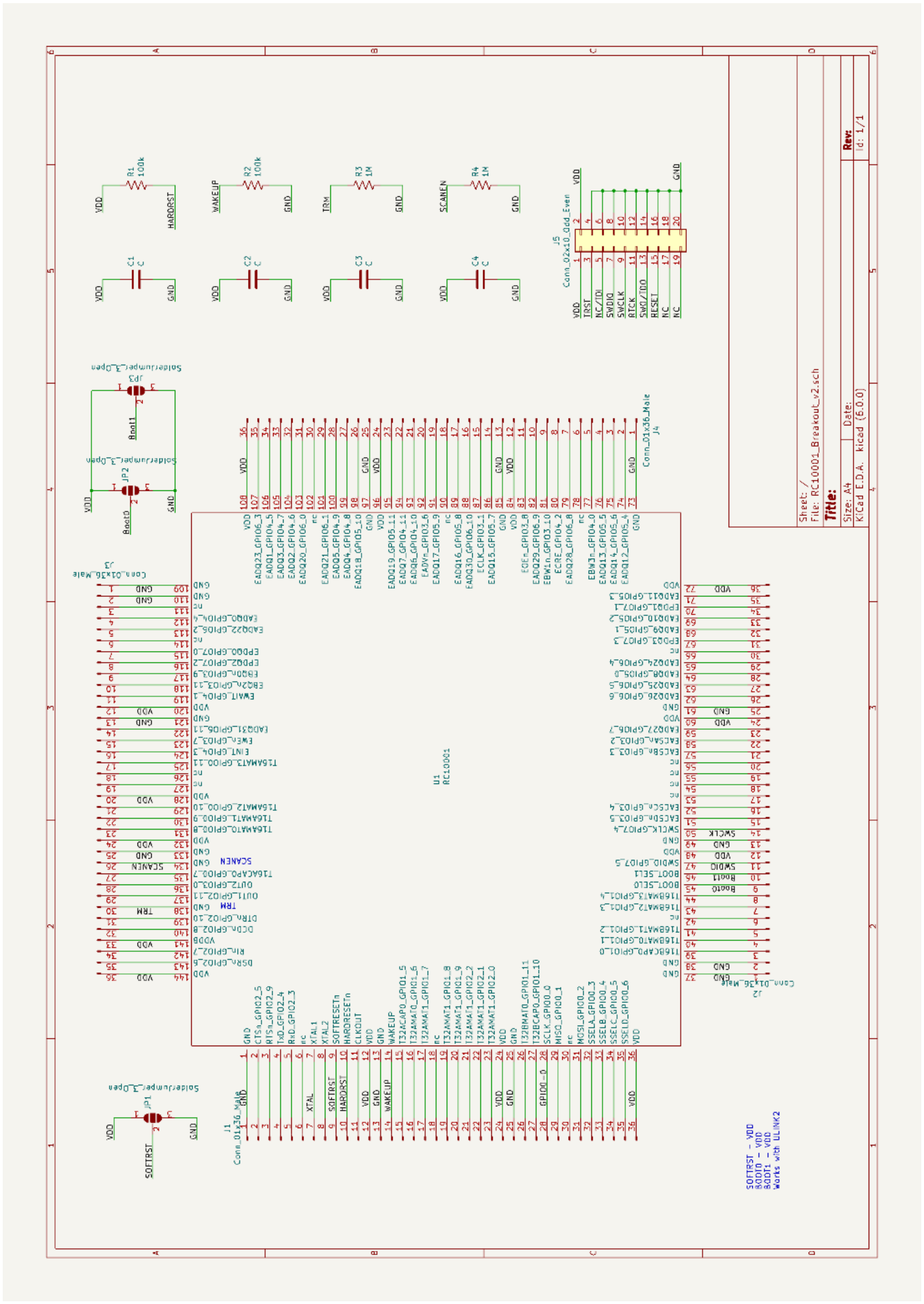


Figure 3. RC10001 microcontroller evaluation circuit.

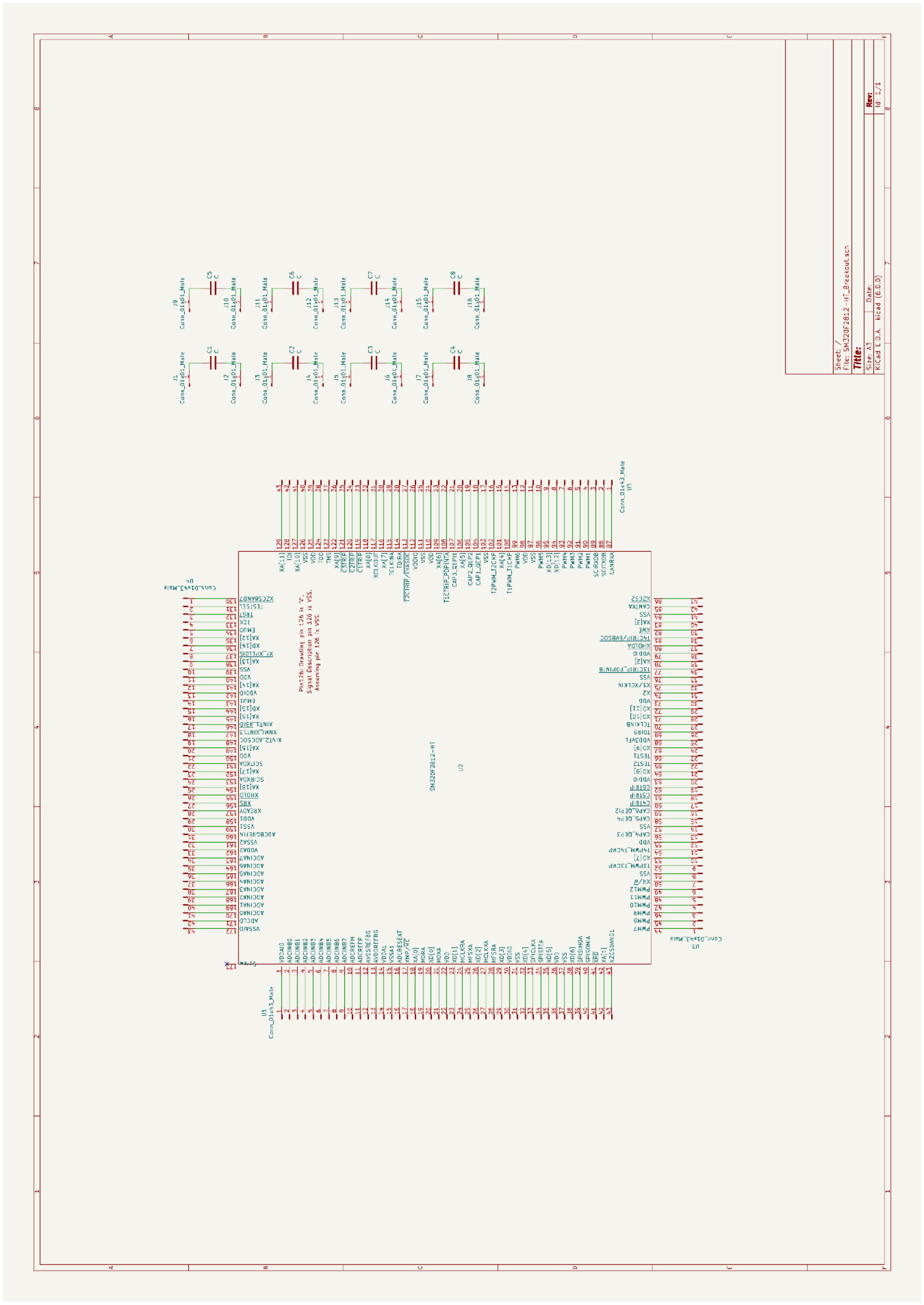


Figure 4. SM320F2812-HT microcontroller evaluation circuit.

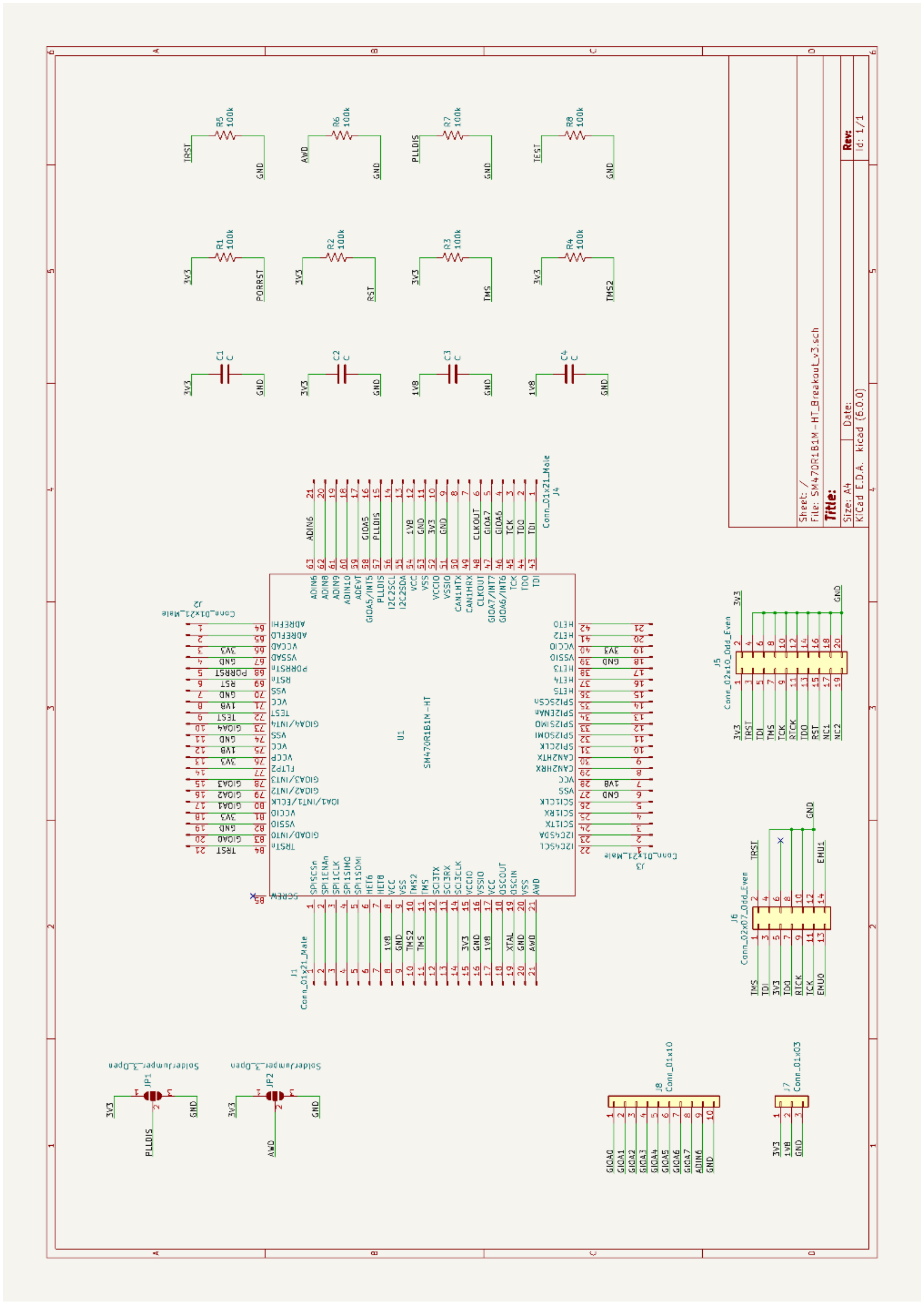


Figure 5. SM470R1B1M-HT microcontroller evaluation circuit.

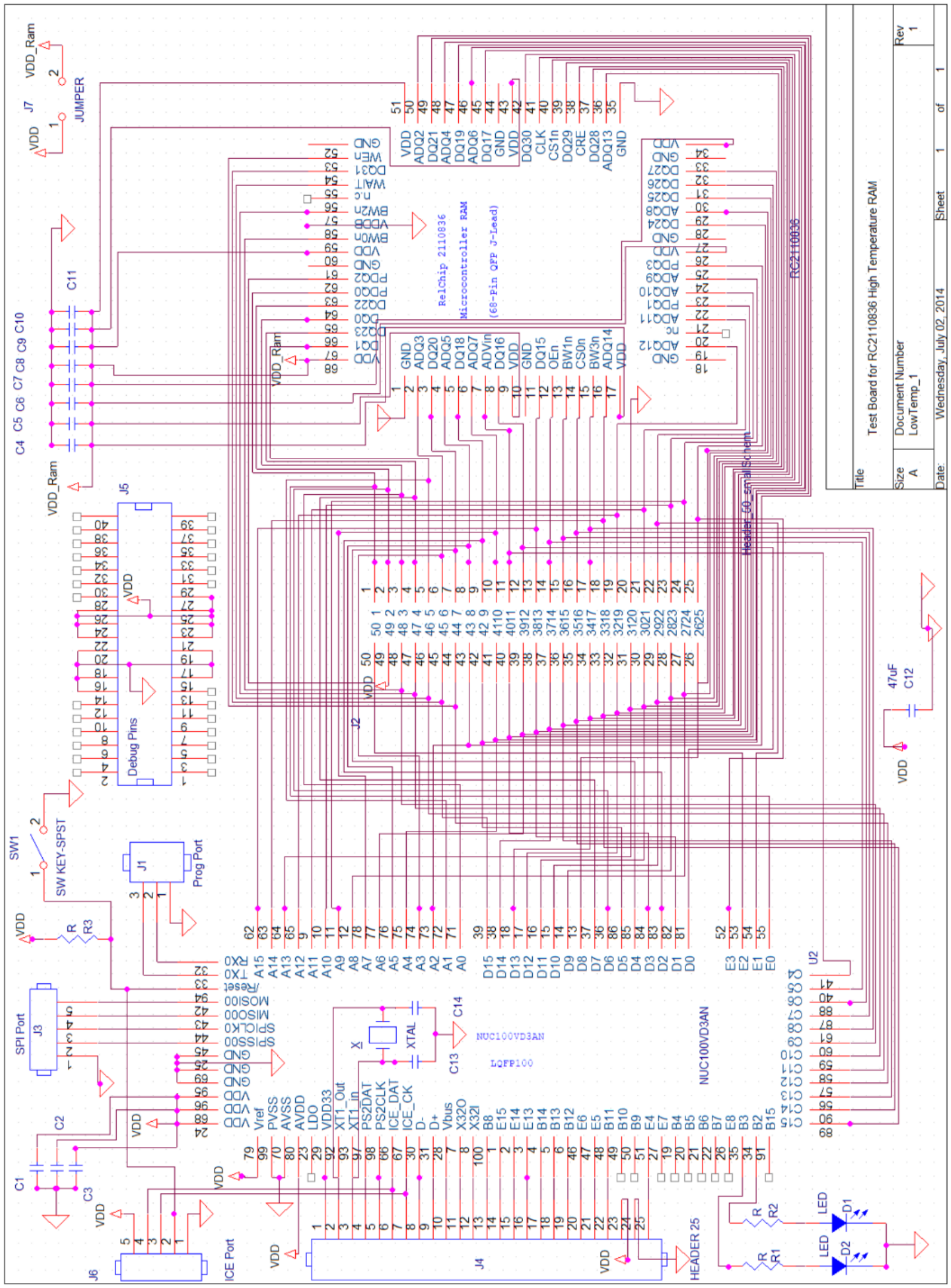


Figure 6. Custom evaluation board for the RC2110836. The NUC100VD3AN is used to interface with the memory chip.

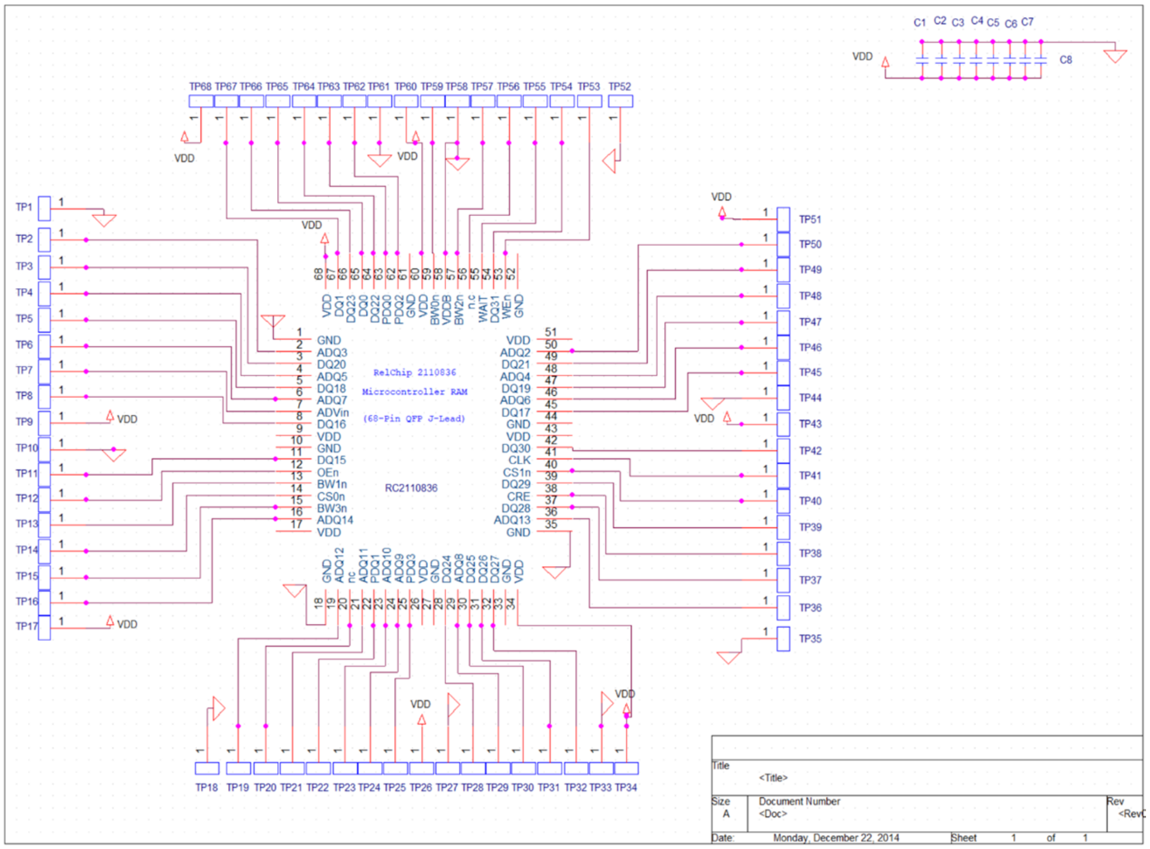

RelChip 2110836
Microcontroller RAM
(68-Pin QFP J-Lead)
RC2110836
VDD
GND
C1 C2 C3 C4 C5 C6 C7
C8
TP1 TP2 TP3 TP4 TP5 TP6 TP7 TP8 TP9 TP10 TP11 TP12 TP13 TP14 TP15 TP16 TP17
TP18 TP19 TP20 TP21 TP22 TP23 TP24 TP25 TP26 TP27 TP28 TP29 TP30 TP31 TP32 TP33 TP34
TP35 TP36 TP37 TP38 TP39 TP40 TP41 TP42 TP43 TP44 TP45 TP46 TP47 TP48 TP49 TP50 TP51
TP52 TP53 TP54 TP55 TP56 TP57 TP58 TP59 TP60 TP61 TP62 TP63 TP64 TP65 TP66 TP67 TP68
Title
<Title>
Size
A
Document Number
<Doc>
Rev
Date: Monday, December 22, 2014
Sheet 1 of 1

Figure 7. Breakout board for the RC2110836 SRAM memory IC.

2.3.2.6. SM320F28335-HT Data Link Circuit

Throughout this effort, the TMS320F2812-HT and SM470R1B1M-HT could not be programmed, and only the RC10001 remained in consideration; however, its lack of a floating-point function and minimal memory made it less than ideal. After discussions with other researchers, the SM320F28335-HT, released around December 2010, was added to the study. The SM320F28335-HT supports MATLAB, which simplifies the programming process, making it the best choice for developing a data link. Programming knowledge and software using SM320F28335-HT may be expanded onto the RC10001 in the future.

The SM320F28335-HT circuit was designed around the datasheet and the TMS320F28335 (sister microcontroller) evaluation board. Considering this chip was used for the data link project, it was more difficult to enable communications across a single conductor wireline compared to the other MCUs. The circuit contains a DAC, amplifier, line driver, crystal oscillator, voltage regulation, and a JTAG port.

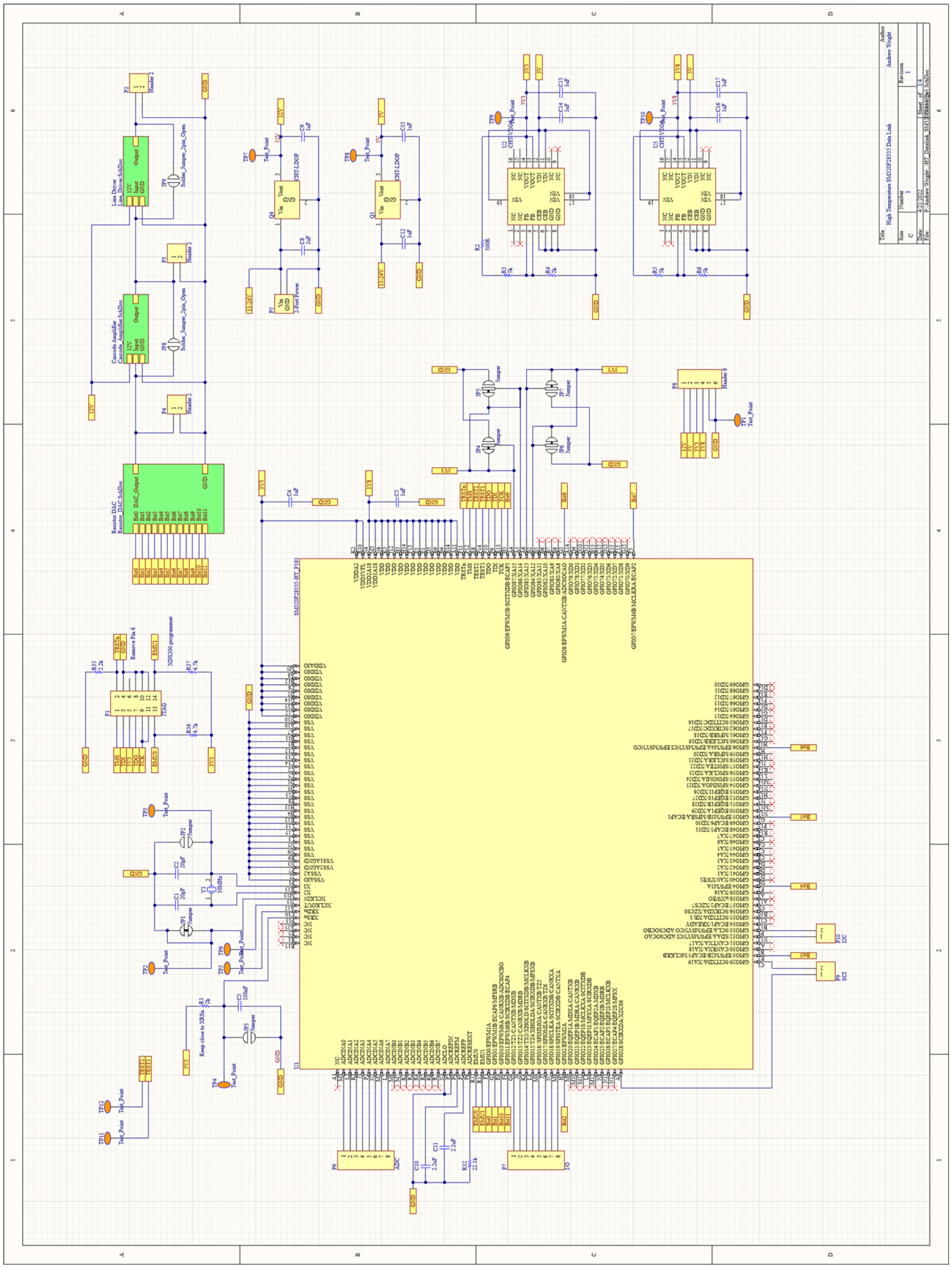


Figure 8. SM320F28335-HT microcontroller and data link circuit.

Figure 9 shows the circuits for the DAC, amplifier, and line driver. The DAC is an array of HT resistors. It operates by setting an output voltage based on one of the 12 pins from the MCU, making it a 12-bit DAC. Its output voltage ranged from 0–3.3V, considering the input/output (I/O) pins operate at 3.3V. To improve the integrity of the transmitted message, the signal did not exceed 1 V. To boost the voltage signal of the DAC, a cascode amplifier was utilized. A cascode typology was used over the common-emitter typology because it does not suffer from the Miller feedback capacitance. With the cascode amplifier, which has a high output impedance, low output impedance is required to drive the low impedance single conductor wireline. This is accomplished using a line driver, a common-drain amplifier typology that can supply the current required. Honeywell's HT N-channel metal-oxide-semiconductor field-effect transistors (NMOSFET) were used to build the amplifier and line driver. Both the line driver and amplifier utilize a NMOSFET on the gate of the input transistor to auto bias the pins as the temperature changes. Ideally, this maintains constant gain profiles as the temperature increases.

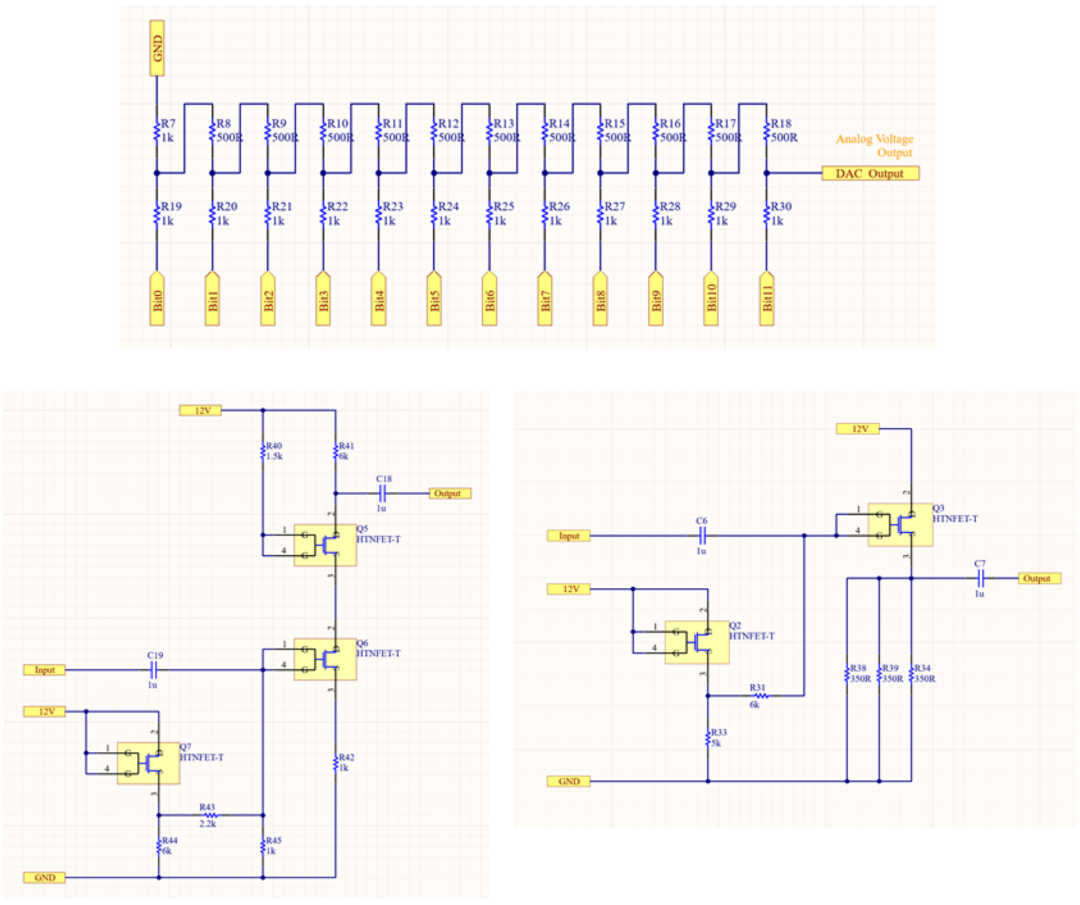


Figure 9. Circuit segment for SM320F28335-HT data link, (Top) resistor-based DAC, (Bottom Left) cascode amplifier, (Bottom Right) common drain-amplifier line driver.

Figure 10 shows the S-parameters for the combined performance of the amplifier and line driver. S-parameters represent gain, reflection, and isolation. In this situation, S21 represents gain, S11 is the input port reflection, S22 is the output port reflection, and S12 is the isolation from the output port to the input port. As shown in Figure 10, the gain profile is about 10 dB from 30 kHz to 180 kHz. Also, the reflection is about -1dB with the 50-ohm impedance ports of the vector network analyzer. Typically, a good reflection is -20 dB, and acceptable reflection is around -10 dB. The performance of the amplifier will need to be improved to minimize reflected energy. In the future, the cascode amplifier will be replaced with an operational-amplifier (opamp) available from Honeywell. Utilizing an opamp will have a higher gain, improve performance at higher temperatures, and lower port reflection profile.

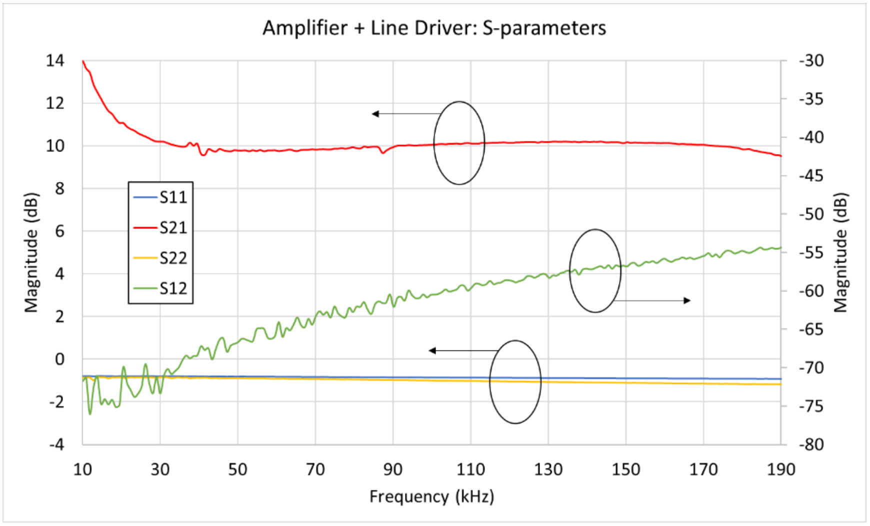


Figure 10. S-parameters for the cascode amplifier and line driver.

Figure 11 shows the insertion loss profile for the 1524 m (5000 ft) single conductor wireline used in this experiment. As seen in the graph, the insertion loss is about 7 dB at 13 kHz and lower. The profile rolls off at 13 kHz. Utilizing the additional gain from the amplifier, the data link was pushed to operate up to 100 kHz where the wireline has an insertion loss of 27 dB. Increasing the bandwidth of the data link output can increase the data rate of the system.

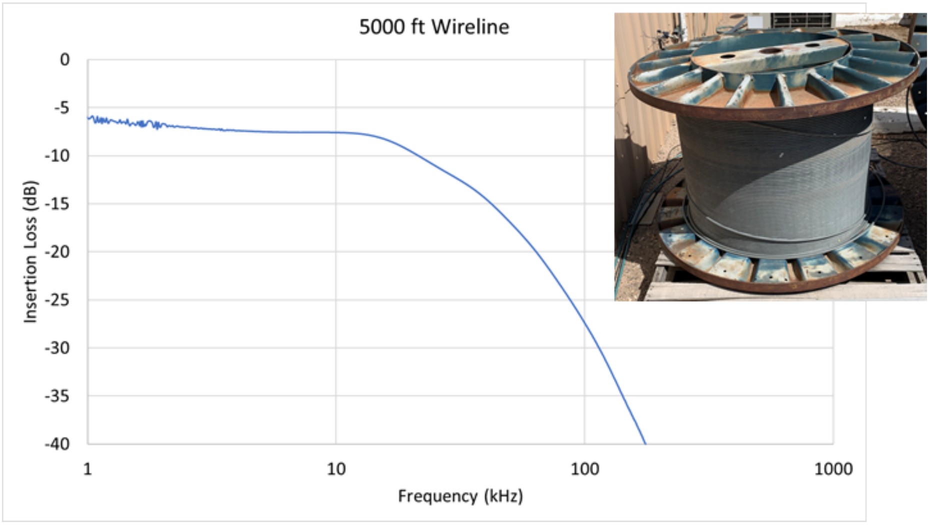


Figure 11. Insertion loss profile of the 1524 m (5000 ft) single conductor wireline.

The data link used the components listed in Table 4. It was designed to operate up to 210°C, the operating temperature of the microcontroller, but is limited by the crystal oscillator. Due to budget and time constraints, it was decided to use the listed crystal. To push the operating temperature further, the crystal can be replaced with an external clock signal.

Table 4. List of Components used on the Data Link

Component	Company	Part Number	Temperature (°C)
MCU	Texas Instruments	SM320F28335-HT	210
NMOSFET	Honeywell	HTNFET	225 (300 1-year)
Crystal	Frequency Management	1931794	200
Resistor	Vishay Dale	ALSR011K000JE12	250 (works at 300)
Crystal Capacitor	Vishay Vitramon	VJ0402D220JXXAJHT	200
Capacitor	Presidio Components	HT1712X7R104J3P1R	250

2.3.3. *Printed Circuit Board Design*

After completing the schematics, a layout used to fabricate the PCB was created for each board, an example of which is pictured in Figure 12. The boards were designed using CAD software such as Altium. In order to mitigate failure at elevated temperatures, the copper traces were placed in the inner layers of the board to prevent them from lifting off. The board was also designed to be bolted to a metal plate to mitigate potential warping at elevated temperatures, hence the 12 screw holes surrounding the IC.

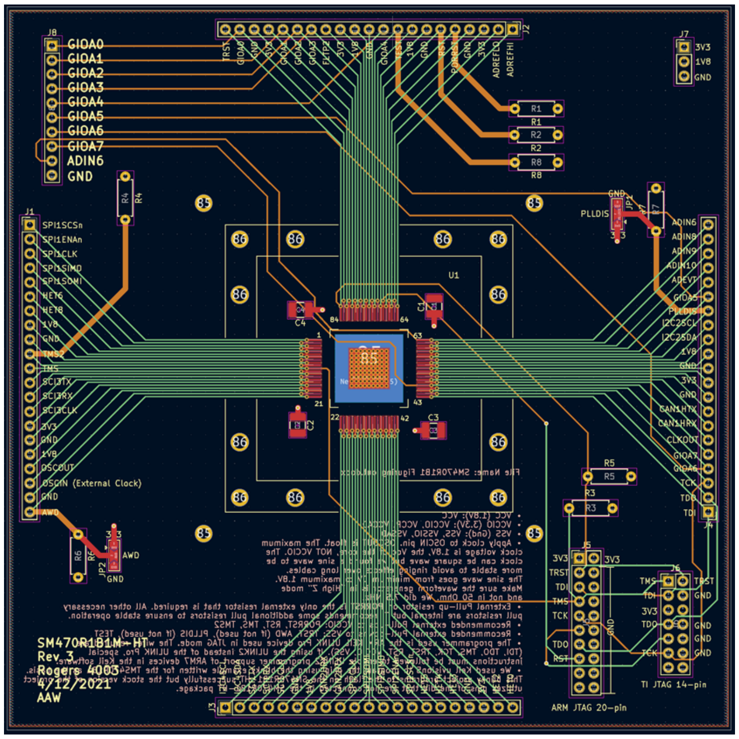


Figure 12. PCB layout for the SM470R1B1M-HT.

With the material and layout defined, the board was fabricated by an external company, Advanced Circuits. Each PCB was relatively simple, containing a total of four metal layers. An important detail selected for the fabrication of the board was to have gold plating on the exposed metal traces. Bare copper will oxidize and tinned pads can degrade the high temperature solder. It was later discovered that solder consumes gold plating at elevated temperatures. The board for the data link was designed during the evaluation of the other microcontrollers. We decided to fabricate the data link board with bare copper plating, accepting that any exposed copper will oxidize over time. Tinned pads is typically standard solder applied to the copper pads from the manufacturer. Standard solder is 60% tin and 40% lead and has a melting point of 190˚C, hereafter referred to as low temperature solder. HT solder used in this research is 97.5% lead, 1.5% silver, and 1% Sn, which has a melting point of slightly above 300˚C. Thus, fusing low temperature solder and HT solder can degrade the temperature performance.

2.3.4. PCB Oven Testing

After receiving the fabricated boards from Advanced Circuits, the boards were evaluated in the oven at elevated temperatures to observe degradation. Several degradation points were observed. The most obvious change is that the board transitioned from a green color at room temperature to a black color at temperatures above 200˚C and later to a white color after sitting at 300˚C for an extended period of time. The boards were coated with a thin non-conductive layer (solder mask) to prevent solder wicking across the metal traces during packaging. Most of the color change results from the solder mask. Figure 13 provides an example of degradation observed at 300˚C, showing the solder mask fractured and delaminated from the Rogers 4003 material. In addition, the Rogers 4003 material was no longer planar; it could not lay flat on the table. Our research found that in multilayer boards, the two Rogers 4003 layers delaminated from each other, creating bubbles between the two layers. This probably resulted from moisture evaporating and causing pressure between the layers, which can be mitigated by baking-out the board at 100˚C for 24 hours.

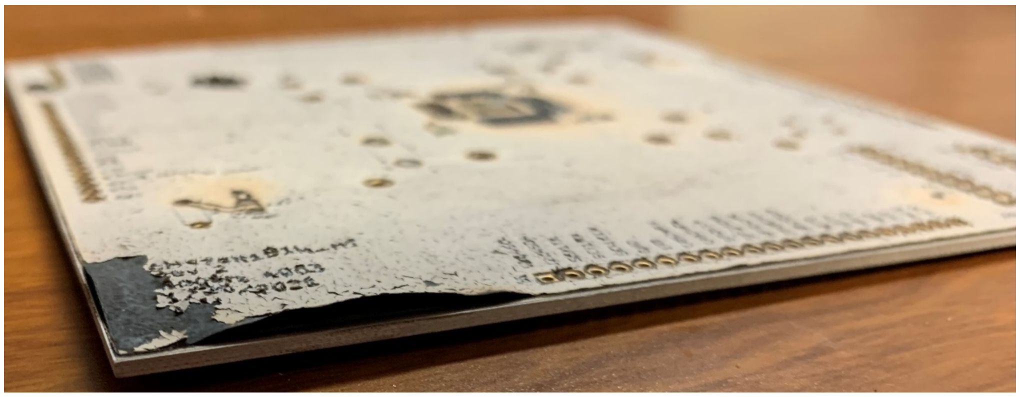


Figure 13. Degradation of PCB at 300˚C. Solder mask changed from a green to a white color and delaminated from the Roger4003 material. The board planarization also degraded and was no longer flat.

The bubbling is prominent in Figure 14, which depicts a cross-section of the PCB. The bubble in the center of the board formed while exposed to elevated temperatures. This can cause significant strain on the solder joints of the IC and PCB, which can lead to the IC debonding.

Figure 14. Rogers 4003 four metal layer board with a bubble formed in the middle of the board after exposure to temperatures above 200˚C. The bubble developed between the two Rogers 4003 layers, the lamination layer.

Figure 15 depicts a PCB with low temperature and HT solder applied to the metal pads. The low temperature solder changed from a silver gloss color to a matte gray color, which indicates the solder is degrading. Degraded solder will typically have higher electrical resistance and/or fracture. The high temperature solder remained a silver color after testing at elevated temperatures, but a black substance appeared surrounding the solder. Notably, the flux was removed from the boards before testing. It was necessary to place the board on a hot plate while using a solder iron to get a relatively decent joint, making it difficult to apply HT solder to the traces. Solder was added to the corner of the large gold metal pad in the middle. As shown in Figure 15, the solder consumed part of the large pad's gold plating.

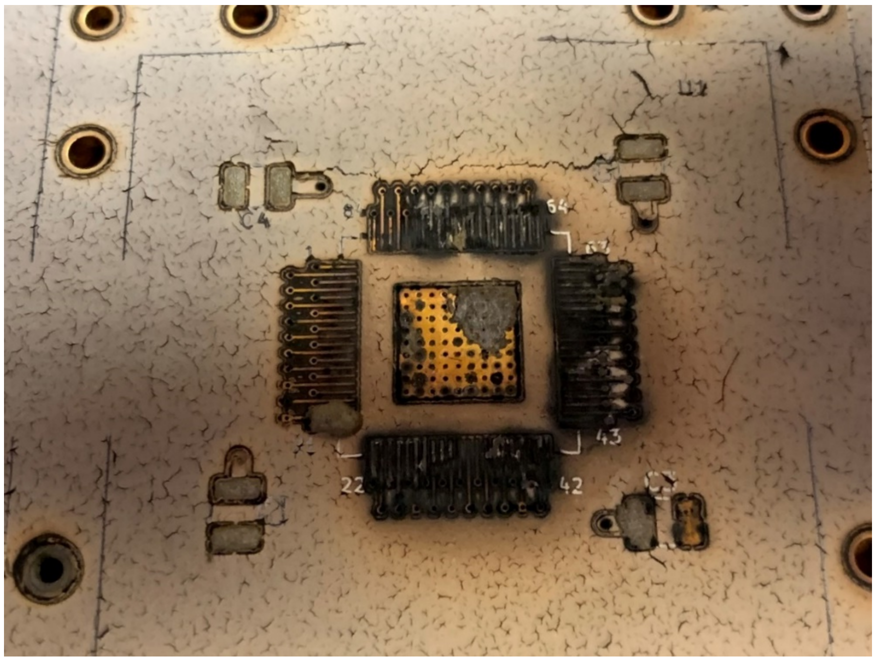


Figure 15. Degradation of PCB at 300˚C. Standard solder and HT solder has been applied to some of the gold-plated traces. Standard solder changed from a silver color to black. The HT solder slightly changed to a textured silver. The solder also consumed the gold plating.

2.3.5. *Packaging and Programming Microcontroller*

2.3.5.1. HT83C51

HT83C51 was the simplest microcontroller to package considering the large spacing between pins, thru hole pins, and small pin count. As shown in Figure 16, the chip was directly soldered to the Rogers 4003 board. HT wires were soldered to the breakout pins on the edge of the board. Packaging details for the HT83C51 are as follows: four metal layered Rogers 4003 PCB with routed metal traces in the inner layers of the board; gold plated pads with 39 mil trace width and clearance; HT solder to attach HT wires; and the thru hole microcontroller. Filter capacitors were not populated (which was not required).

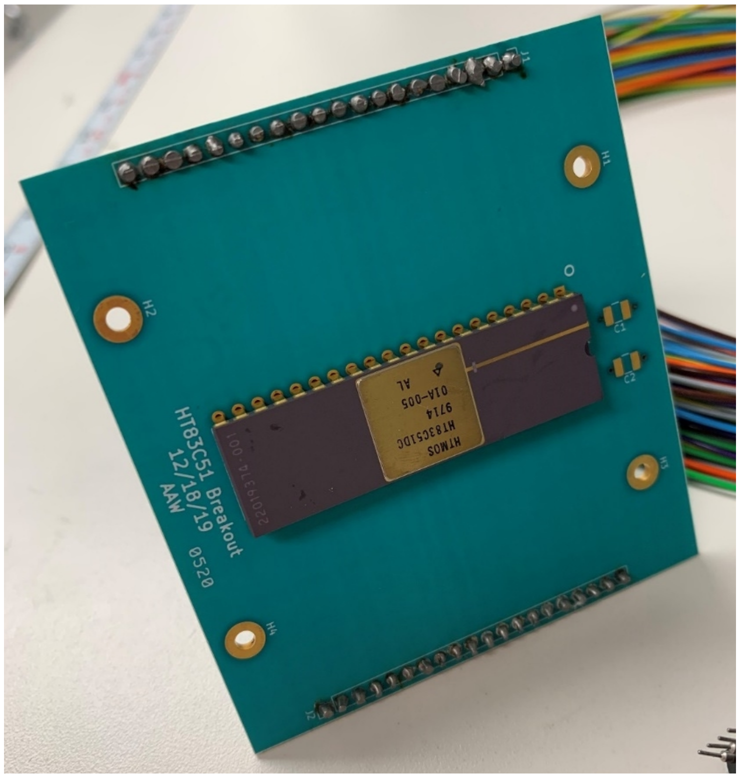


Figure 16. HT83C51 soldered to Rogers 4003 breakout board with HT wires.

The evaluation board for the HT83C51 microcontroller is shown in Figure 17. The evaluation board was custom made by SNL over a decade ago. It contains memory, power regulation, and custom firmware to program the HT83C51. Honeywell did not supply hardware to program the HT83. To use the microcontroller, SNL developed technology to program the device. One method developed to program the HT83C51 involved using a field programable gate array (FPGA). SNL developed firmware for the FPGA to enable communications between the computer and the HT83C51.

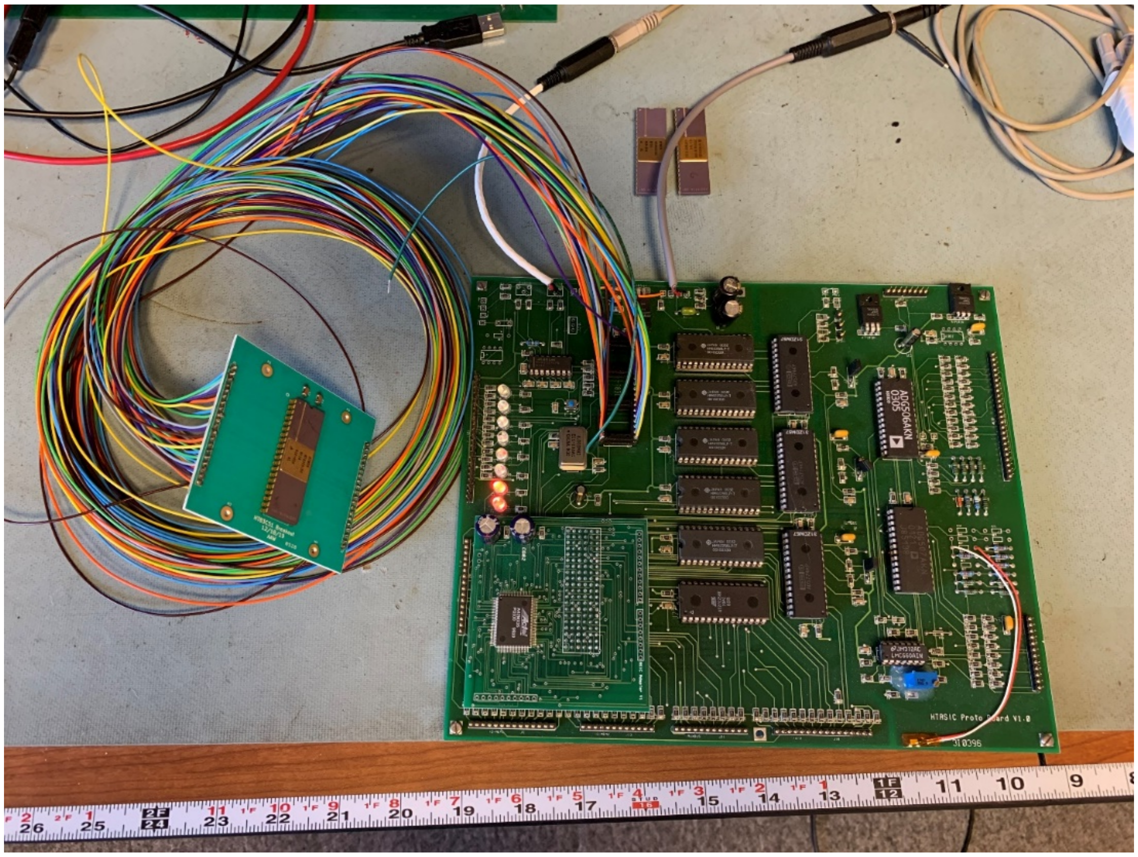


Figure 17. Custom SNL evaluation board for the HT83C51. Utilizes an FPGA with custom code to program the microcontroller. Evaluation board also includes memory and power regulation.

After developing the FPGA and firmware, SNL enabled the programmer technology to operate at the same temperature as the HT83C51 by converting the firmware code into a transistor format. An effort was then put together to fabricate the IC utilizing Honeywell's SOI HT IC fabrication technology. This enabled the ability to program the HT83C51 while the entire board is exposed to elevated temperatures deep within a borehole. Figure 18 shows the fabricated programmer for the HT83C51 chip.

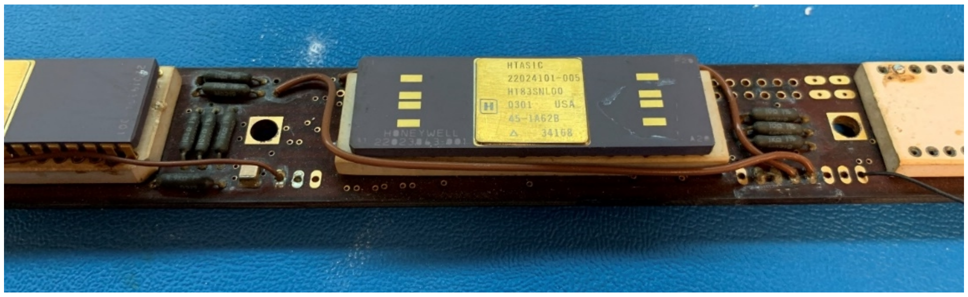


Figure 18. Custom SNL HT application specific integrated circuit (ASIC) programmer for the HT83C51. Utilized for programming the microcontroller while operating at elevated temperatures.

The breakout board is essentially an extension cord from the evaluation board to the IC. The evaluation board remains at room temperature, while the IC is exposed to high temperatures. The issue that occurred with this approach is that the microcontroller could not be programmed properly. The long wires either caused noise or a timing inaccuracy even though all the wires were made to be the same length. To bypass this issue, the breakout board was connected directly to the evaluation board as shown in Figure 19. Once the chip was programmed, the breakout board was removed, and the software continued to operate as expected. This was not ideal for evaluating the IC, but due to budge limitations, it was enough to compare to the other microcontrollers. The software used was simple, the MCU constantly transmitted a digital string of characters, "Hello World!" If the message stopped, either the MCU became corrupted and required reprogramming or the MCU was no longer functioning. The code used to evaluate the microcontroller is listed in Appendix A.1.

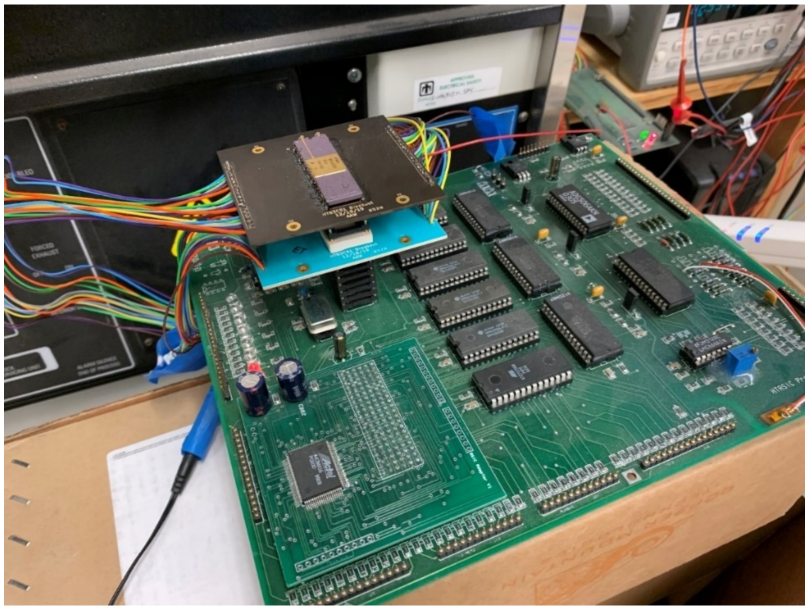


Figure 19. HT83C51 in the process of programming with the development board.

2.3.5.2. RC10001

The RC10001 was initially soldered to the Rogers 4003 PCB with low temperature solder. Low temperature solder was chosen because it only needed to operate for one month and would not be exposed to vibration. This chip was evaluated at 300˚C, thus the solder would be in a liquid state. Ultimately, the chip was soldered with HT solder to mitigate packaging failures observed in Section 2.4.3. The board consists of a four metal layered Rogers 4003 PCB with routed metal traces in the inner layers of the board as well as gold plated pad with 6 mil clearance for the IC pads and 9 mil routed trace clearance. To program the device, we used Keil software and a Keil Ulink2 programmer. Steps to program the device are listed in Appendix A.2. the code used for the RC10001 was even simpler than the HT83C51 software. Building off the RC10001 example code, the microcontroller pulsed an I/O pin continuously. Evaluation code used for the oven testing is listed in Appendix A.3. Only four wires were soldered to the board as shown in Figure 20: two for power, one for an external clock, and the last for the I/O pin, which was to be observed during testing.

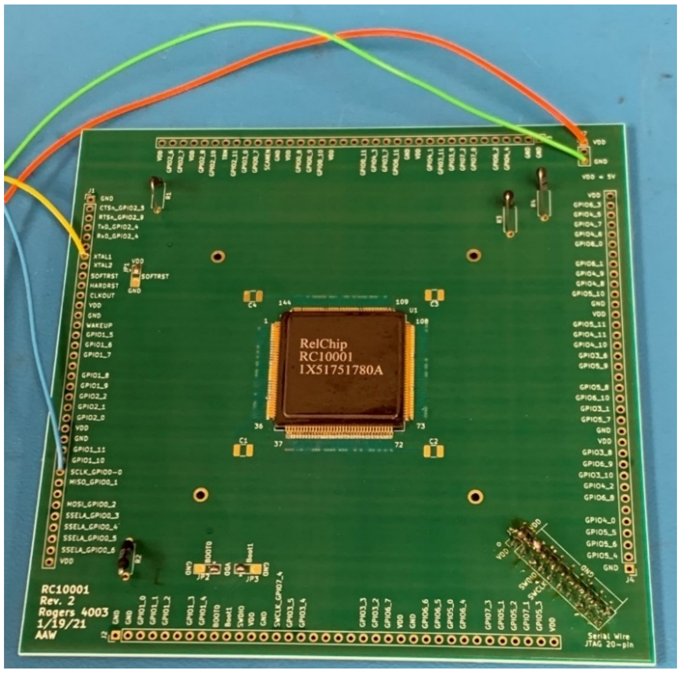


Figure 20. RC10001 soldered on Rogers 4003 PCB with HT wires.

2.3.5.3. SM320F2812-HT

Board details for the TMS320F2812-HT are as follows: four metal layers with a majority of connection traces in the inner layers; gold plated pads, 7 mils trace width and clearance; and Rogers 4003 material. A picture of the packaged IC with the PCB is shown in Figure 21. The board has connections for the HT wires and the microcontroller. Resistors were soldered to the breakout pins on the edge of the board to properly set the microcontroller. TMS320F2812-HT was soldered with low temperature solder to the Rogers 4003 PCB. The microcontroller is a surface mount device, but it has a rail around the pins. The rail can be removed, but the team decided to leave the rail attached and flipped the chip in reverse. The rail was left for the packaging aspect because of the additional bonding strength from the solder. In additional, the rail's mounting hole points can be used to attach screws that strengthen the mechanical structure between the IC and PCB.

Unfortunately, after the effort of packaging the device, no attempt allowed the user to program the microcontroller. Many approaches were attempted to program the device. Firstly, Code composer 10 with the XDS200 programmer was used but with no success. This was followed by replacing the XDS200 with the XDS100v2, XDS510, or XDS510 plus, but the IC was not detected. All the solder connections were triple checked by two electronics engineers as well as ensuring proper settings and JTAG connections. Later, the Spectrum Digital eZdsp TMS320F2812-HT evaluation board was purchased. This evaluation board contained the sister chip of the SM320F2812-HT IC. It also contained the TMS320F2812-HT microcontroller, which is the low temperature version of the IC. The same programming approach was used again, this time on the evaluation board. When that failed, Code Composer 3 was used in an attempt to detect the device. Again, the software could not recognize either the package device or the evaluation board. The effort attempted two final approaches: an older version of the Windows OS was used and the parallel port on the evaluation board was used with an old computer to try and detect the device. With all attempts failing to program the device, the SM320F2812-HT was abandoned under this effort and was not evaluated at elevated temperatures.

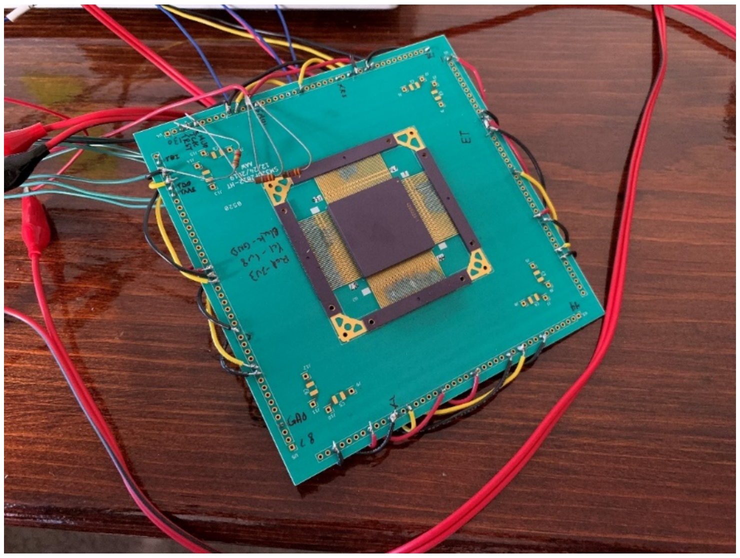


Figure 21. SM320F2812-HT 172-pin soldered on Rogers 4003 PCB.

2.3.5.4. SM470R1B1M-HT

Board details for the SM470R1B1M-HT are as follows: four metal layers with a majority of the connection traces in the inner layers; gold plated pads, 7 mil trace width and clearance; and Rogers 4003 material. The package is pictured in Figure 22. The board has connections for the HT wires, the surface mounted microcontroller, eight HT resistors, and the JTAG header. SM470R1B1M-HT was soldered with low temperature solder to the Rogers 4003 PCB.

To program the device, ULINK Pro and Keil software was used. TI Code Composer could not be used because the IC is no longer supported by the software. The IC was released when Code Composer 3 originally came out, like the TMS320F2812-HT. Keil was used to program the device because it had a library for the TMS470R1B1M (TMS470) IC, the low temperature version of the IC. Initially, it was challenging to properly configure the external resistors to the IC. As such, the TMS470R1B512 (similar low temperature IC) evaluation board was initially used to confirm detection of the chip with the Keil software. After confirming detection of the TMS470 and properly configuring the SM470R1B1M-HT, the Keil software successfully detected the SM470R1B1M-HT. In an attempt to program the device, the team ran into an issue. Even though the TMS470 was like the SM470, there were differences that hampered the ability to easily program the SM470R1B1M-HT. The TMS470 has features such as additional I/O blocks. Even when trying to program the SM470R1B1M-HT with the TMS470 library, the registry to configure the I/O port would not work, even though the Keil software would note the chip was programmed properly. To program the IC, a library would need to be handwritten to define all the registries of the IC, something the project budget could not support. Later it was discovered that another software, IAR Systems, had the library set to program the SM470R1B1M-HT specifically. Unfortunately, considering the cost, the budget could not support the software's purchase. Near the end of the effort, an external company successfully programmed the device, indicating the IC could potentially be used in HT applications; however, considering SNL could not program the device at the time, it was not evaluated under this effort.

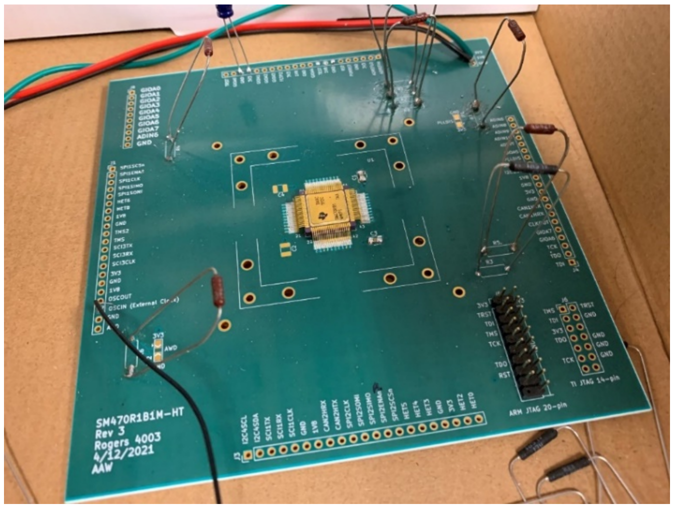


Figure 22. SM470R1B1M-HT soldered on Rogers 4003 PCB.

2.3.5.5. SM320F28335-HT

Unlike other microcontrollers, the SM320F28335-HT was also used for evaluating a data link for communications across 1524m (5000 ft) of single conductor wireline as discussed in Section 2.3.2. Population of the PCB is shown in Figure 23. The microcontroller is the large gray IC with a gold square pad in the middle. Considering it would only be evaluated 210˚C, this circuit was built on a Rogers 4003 PCB and bonded using low temperature solder. Board details for the SM320F28335 are as follows: four metal layers with majority of the connection traces in the inner layers; bare copper pads; and 10 mils trace width and clearance.

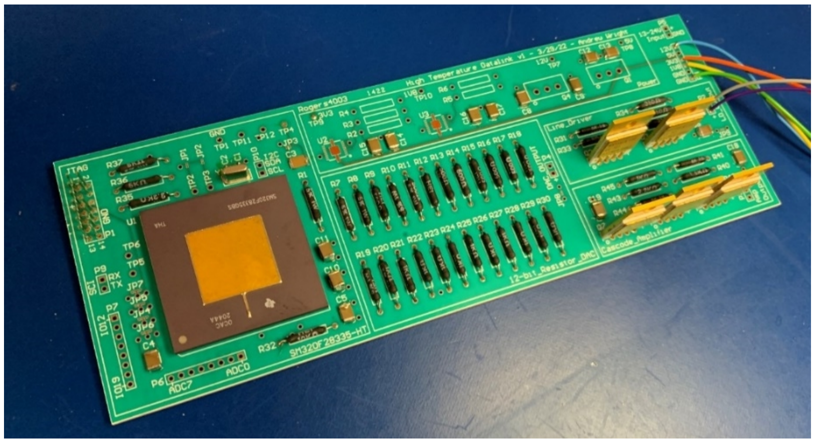


Figure 23. SM320F28335-HT soldered on Rogers 4003 PCB. PCB designed for data link application. Incorporates HT resistor DAC, HT amplifier, and HT line driver. Design also includes power regulation, but components were not utilized.

Along with a more complex PCB, the SM320F28335-HT used a more complicated software compared to the other devices. Like the other devices, the code continuously output a set of data; however, complexity arose when the digital message was converted to an analog signal using keying techniques. As noted earlier, the data link is designed to send messages at high-speed across a single conductor wireline. There are several manipulations that can occur to a sinusoidal signal to represent a digital bit. These can include turning on and off the sine wave, changing the frequency, amplitude, or phase. Keying techniques are shown in Figure 24. By combining these techniques, data rates can be increased significantly. As reviewed earlier, a previous effort utilized OFDM+QAM which is the manipulation of frequency, amplitude, and phase with the high resolution and wide frequency bandwidth of the wireline. The goal was to duplicate the effort but replace the National Instruments DAQ with an HT MCU, but ultimately the code was simplified and used OFDM+ binary phase shift keying (BPSK). OFDM+BPSK is manipulation of the frequency and only two-phase shifts. As such, this has a much lower data rate compared to previous efforts. This slower technique was used because it would be challenging to convert the original MATLAB code into C code for the microcontroller. The effort determined it would be better to start simple and rewrite the MATLAB code using the built-in MATLAB BPSK libraries.

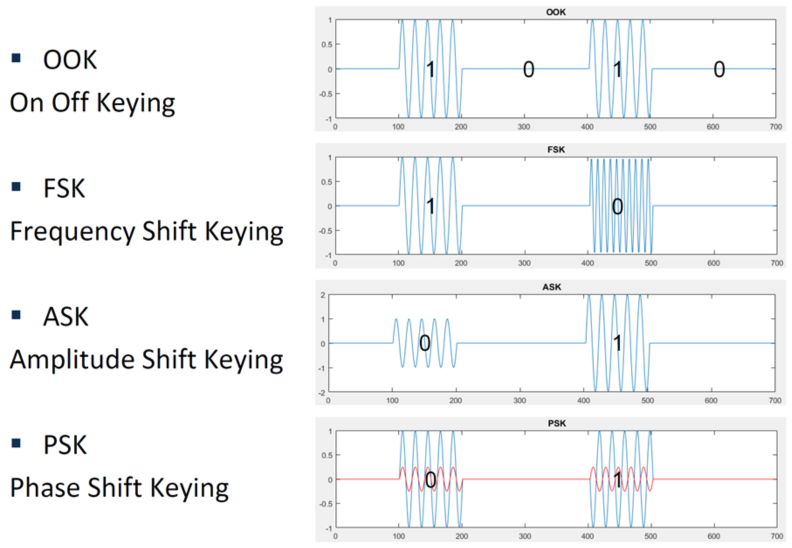


Figure 24. Illustration of the basic keying technique for transmitting data.

Johnny Silva from SNL made the basic OFDM+BPSK MATLAB code as a proof of concept on the computer and demonstrated the capability to convert it to C code using the built-in MATLAB tools. This was then transferred to Francis Tiong from MathWorks to optimize the code and get it to operate on physical hardware. A procedure was written describing how to use the XDS200 programmer, MATLAB, and TI Code Composer 10 to program and alter settings for the SM320F28335 data link, which can be found in Appendix A.5. Partial code for the data link can be found in Appendix A.6.

For the experiment, the code transmitted the message "Live long and prosper, from the Communications Toolbox Team at MathWorks!" The microcontroller would then repeat the message continuously. To do this, the message was transposed into the OFDM+BPSK format. This was outputted from the microcontroller to its 12 I/O pins. Those 12-bits were then converted to an analog signal with the resistor DAC, which was amplified and propagated through the 1524 m (5000 ft) single conductor wireline. The signal was then captured by Pico Technology's PicoScope oscilloscope. The data was saved in a MATLAB format with the PicoScope software, after which the signal was converted back to the original message using a MATLAB script. Along with that, the MATLAB script plotted a constellation diagram, which represents a digital modulation scheme. Shown in Figure 25 is a sample constellation diagram. There are four blue dots (symbols) on the diagram representing a quadrature phase shift keying (QPSK) constellation, to which each symbol represents 2 bits. The larger the constellation, the larger number of bits that can transmit per symbol.

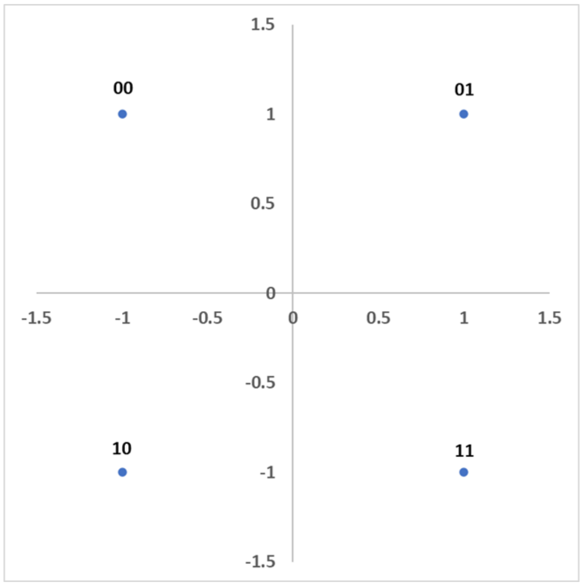
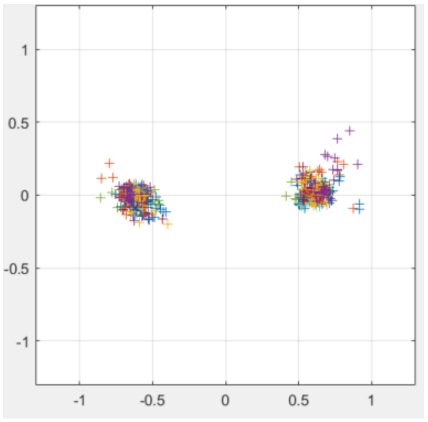
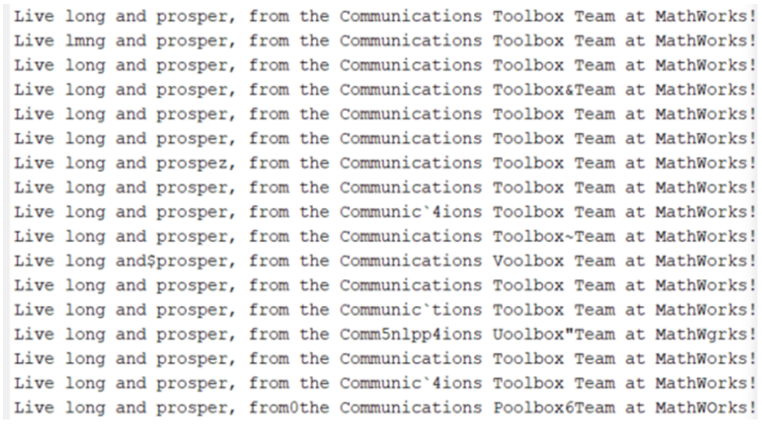


Figure 25. Sample constellation diagram for a 4-QAM.

A constellation diagram is important because it can show interference that may have occurred when the signal was first generated to the receiver. There are many ways a signal can degrade, including random EM signals in the air, motion to the cables, power supply noise, thermal noise, and even radiation from the sun.

Figure 26 shows the received signal's constellation diagram and the messages received. On the left is the constellation diagram for the messages received by the oscilloscope. Considering the technique used is BPSK, there are only two symbols in the diagram. Each symbol represents a single bit. The symbols are not a single point on the diagram, instead the points are spread around the expected point on the diagram. This is due to noise the data link and wireline experienced. The points are in the general region within the acceptable threshold defined by the MATLAB script. The signal was then converted to the messages shown in the image to the right. When there is noise on the signal, it can degrade the message, hence the reason for the incorrect characters seen in some of the messages. There are multiple ways to improve the signal integrity, which will be explored in future efforts.

```
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live lmng and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox&Team at MathWorks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prospez, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communic`4ions Toolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox~Team at MathWorks!
Live long and$prosper, from the Communications Voolbox Team at MathWorks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communic`tions Toolbox Team at MathWorks!
Live long and prosper, from the Comm5nlpp4ions Uoolbox"Team at MathWgrks!
Live long and prosper, from the Communications Toolbox Team at MathWorks!
Live long and prosper, from the Communic`4ions Toolbox Team at MathWorks!
Live long and prosper, from0the Communications Poolbox6Team at MathWOrks!
```

Figure 26. Received data on the oscilloscope, transmitted from the data link. (Left) Image of constellation plot of the BPSK signal and (Right) messages that were received.

2.3.5.6. RC2110836

Board details for the RC2110836 are as follows: four metal layers with a majority of the connection traces in the inner layers; gold plated pads, 12 mils trace width and clearance; and Rogers 3003 material. A picture of the package IC with the PCB is shown in Figure 27. The board has connections for HT wires, the surface mounted microcontroller, and filtering capacitors. The IC was soldered with low temperature solder to the Rogers 3003 PCB.

The NUC100VD3AN microcontroller was used to set each register to a 1 or 0 bit on the RC2110836 SRAM. The MCU would then read each register and transmit the data to the computer. The computer then displayed and saved the data as a hex value, representing the bits of the registers. After all the registers were set and data saved, the microcontroller then flipped the bit and repeated the process. The flipping of the bits repeated continuously. If any register failed to flip, the hex value would change, indicating to the user that a register had failed. MATLAB software on the computer was used to interact with the microcontroller. The code is shown Appendix A.4.

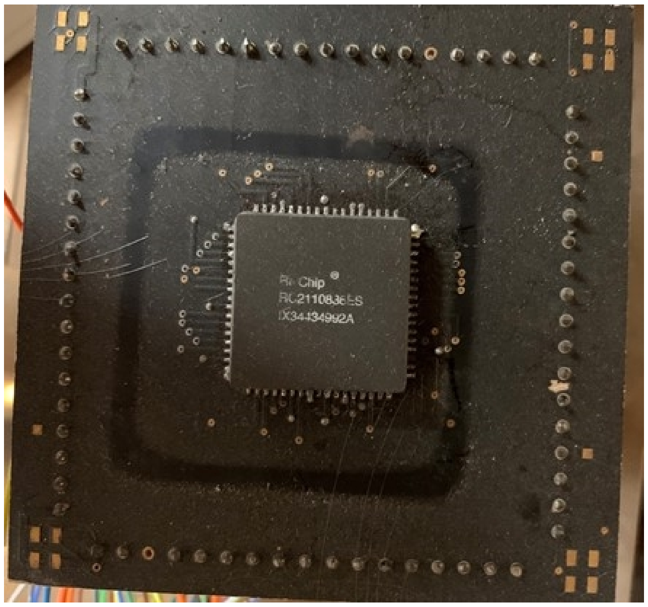


Figure 27. RC2110836 low temperature soldered to Rogers 3003 PCB with HT wires. Pictures of the packaged device were not taken before the oven evaluation. Originally the board was a green color.

2.4. Evaluating Packaged Microcontrollers and Memory

The four ICs were evaluated considering factors from the previous section. This includes the HT83C51, RC10001, RC2110836, and the SM320F28335.

2.4.1. Evaluating HT83C51

The HT83C51 board was evaluated in an oven reaching 300˚C. Altogether, we evaluated two packaged HT83C51s. The first package operated at 200˚C for 67 hours with no issues. After the oven temperature was increased to 300˚C, the IC operated for 6 hours before failure. The oven was deactivated and left to cool after which the chip was power cycled and reprogrammed. The device operated again, outputting the "Hello World!" message, but as the temperature of the oven was slightly increased, the IC failed again. The evaluation determined that the IC can only operate at room temperature. A second IC was packaged, and the temperature was increased slowly to collect more data. This time, the PCB was baked-out at 100˚C for over 24 hours to remove any moisture from the board. Testing started at 200–215˚C for 125 hours with no observable errors. Next, the temperature was increased to 225–235˚C for 206 hours, 250˚C for 118 hours, and 255–265˚C for 50 hours with no observable issues. Finally, the temperature was increased to about 275˚C, after which the IC failed at 12.8 hours. In the datasheet for the IC, it is designed to operate at 225˚C and is capable of one year of operation at 300˚C. As the results show, one of the evaluated ICs operated at 300˚C for only 6 hours. Further testing is required to understand why the ICs failed prematurely. One unlikely possibility is the age of the IC. The ICs evaluated were fabricated over 10 years ago, so there might be a chance the transistors of the IC degraded by sitting on the shelf. During testing, we monitored the current and observed the IC drew about 38 mA at room temperature and 30 mA at 300˚C. Typically, current increases with elevated temperatures, but with the HT83C51, the current decreased. After the packaged device was exposed to temperatures of 300˚C, several degradations were observed on the board. The green solder mask changed to a white color as shown in Figure 28. One can also see fractures lines occurring above the routed traces between the IC and breakout pins. It is unclear why the traces encouraged that effect considering they are within the inner layers. The HT solder has blackened as previously shown in Section 2.3.4. Finally, the text on the top of the IC (gold region) faded away.

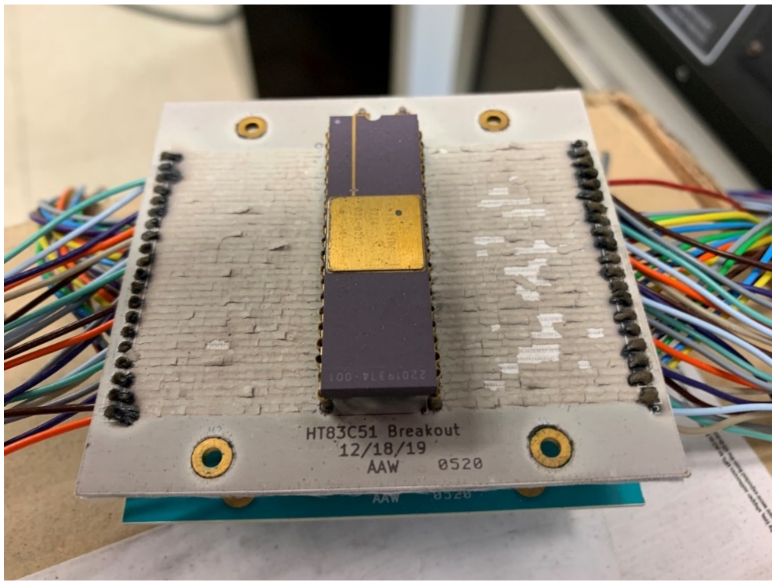


Figure 28. HT83C51 on the breakout board after 300˚C evaluation. The board transitioned from a green solder mask to a white color, forming various fractures.

2.4.2. *Evaluating RC2110836*

During the test, a cylindrical metal weight was placed on top of the memory IC to mitigate delamination from the PCB. A second hollow metal cube was placed on top of the PCB to help mitigate deformation of the PCB near the IC. At room temperature, 200˚C, and 300˚C, the current draw of the chip was 2.5 mA, 3.5 mA, and 21 mA, respectively. During evaluation of the IC at 300˚C, the packaging failed after the IC debonded from the PCB as a result of strain from the deformation of the board and the degradation of the low temperature solder. Considering the challenges associated with HT solder, the same chip was resoldered with low temperature solder. Following the repair, the board continued the evaluation. The packaged device operated at 300˚C for a total of 528 hours with no additional failures. To push the IC further, the temperature was increased to 305˚C, where the device operated for another 816 hours before failing. The packaged microcontroller, with and without the weight after the evaluation in the oven, is pictured in Figure 29. The board started as a green color, changing to a white color after hundreds of hours at 300˚C. The surface of the board contained many fractures and visibly warped around the edges.

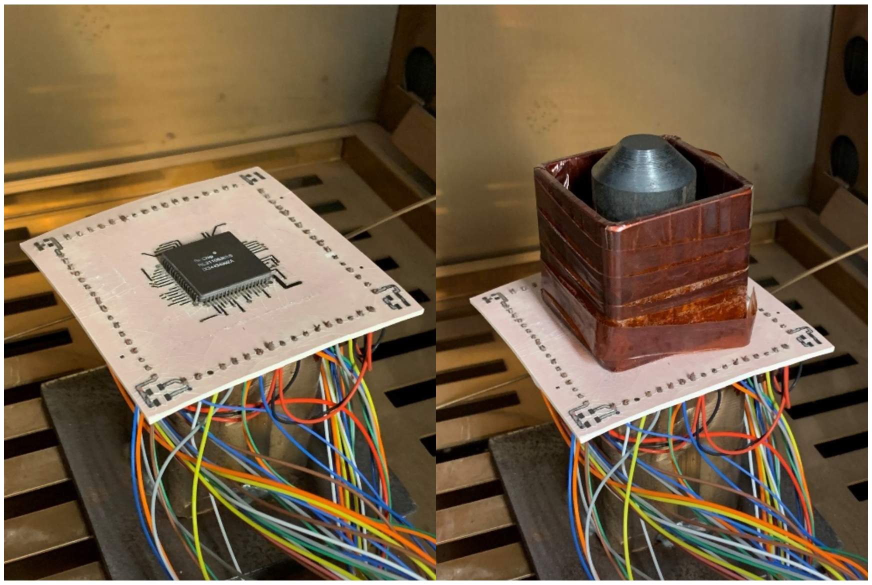


Figure 29. RC2110836 and packaging after evaluation at 300˚C for over a month. Pictured on the right is the microcontroller with a weight on top to mitigate IC deformation.

2.4.3. Evaluating RC10001

One of the most challenging HT ICs evaluated under this effort was RC10001 due to it being a surface mount device (SMD) as well as various issues associated with the PCB's exposure to high temperatures. With the packaging techniques we had available, a total of three packages were assembled to obtain the best results in the allotted time. Following the success of testing the SRAM IC, the first RC10001 was soldered with low temperature solder due to the difficulty of working with HT solder on a tight pin pitch on the IC. The set was exposed to temperatures above 200˚C for 42.5 hours without failure until the temperature reached 276˚C. The chip failed after de-bonding from the board. As described in Section 2.3.4, the board bubbled and the solder degraded. As shown in Figure 30, the solder degraded and consumed the gold plating after being exposed to elevated temperatures for an extended period. Two pins in the image were soldered while the other pins were not. The gold plating on the pins was completed stripped off at elevated temperatures. Along with these challenges, there was a coefficient of thermal expansion (CTE) mismatch between the IC and PCB. CTE mismatch occurs when there is a delta in physical expansion between two materials as the temperature increases. As the CTE mismatch increases it causes strain on the solder joint, and the IC physically removed itself from the board.

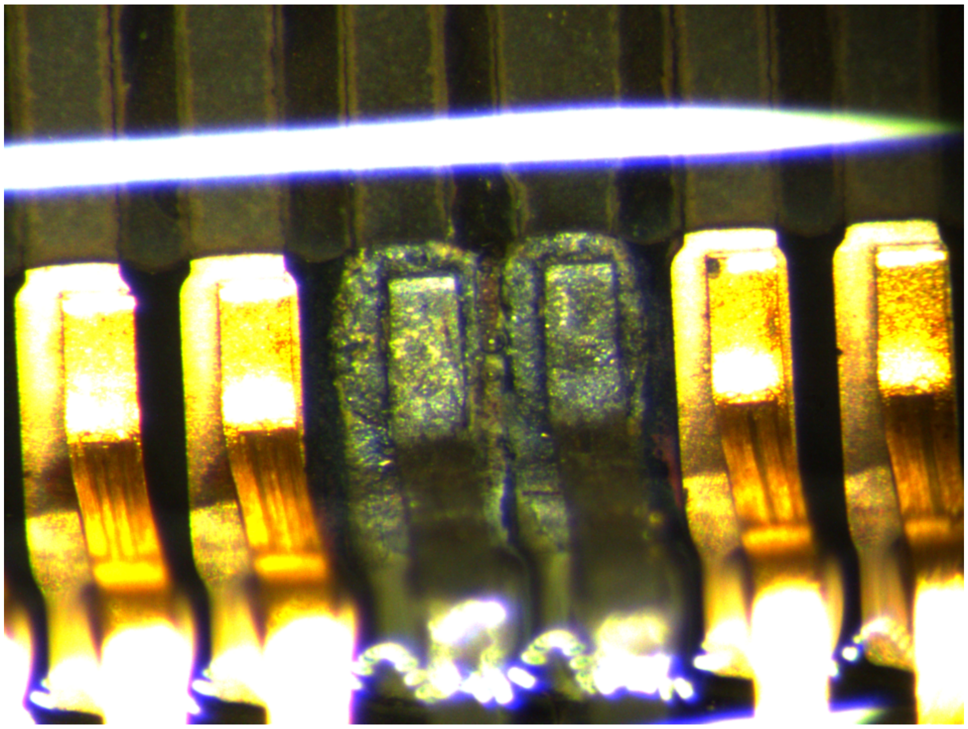


Figure 30. RC10001 connection pins. The center two pins are soldered, while the remaining gold-plated pins were left unsoldered. At elevated temperatures, the solder consumed the gold plating off the two pins.

Due to budget constraints, the same board design was utilized (ten duplicate PCBs were fabricated), and low temperature solder was used again. This time, a large metal weight was placed on top of the microcontroller as shown in Figure 31. Ideally, the weight would mitigate deformation of the PCB. With a new IC and PCB, the board was placed in the oven at 100˚C for four hours to remove any moisture. The temperatures were then increased to 256˚C, where it failed after 2.4 hours. After failing, the board was left in the oven. Power to the board was left on and the temperature was increased to 300˚C, where records show the package's power lines shorted. The IC was exposed to those temperatures for 19 hours before being removed. After it was removed from the oven, the packaged device was evaluated under the microscope. The chip did not delaminate from the PCB, but it appeared the solder slightly expanded between the pins, potentially causing digital and power line shorts, resulting in the initial failure at 256˚C. The power line shorted as the board was sitting at 300˚C. The expanded solder was cleaned with a micro chisel. After cleaning the solder, the short remained. Thus, the IC was removed from the board to investigate further. Even after removing the chip and cleaning all the solder, the short could not be found, indicating that it most likely occurred within the inner layers.

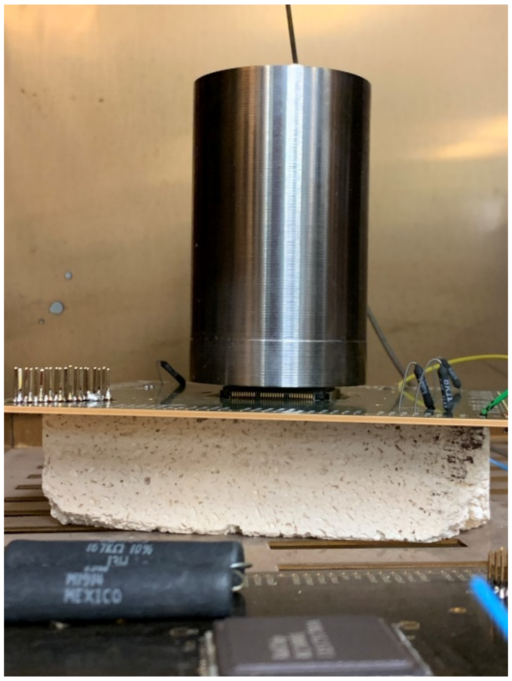


Figure 31. RC10001 microcontroller with a metal weight on top to mitigate deformation under elevated temperature evaluation.

To mitigate the issues observed in the second package, the third package used HT solder. It is challenging to use HT solder on the RC10001 when processing by hand, so to simplify the process, only the necessary pins were soldered. The board had to be placed on a hot plate to use the HT solder. To focus heat on the chip, a small block slightly larger than the IC was placed below the board and aligned with the chip as shown in Figure 32. The hot plate temperature was increased to 210°C and a special soldering iron that flows inert gas (nitrogen) over the iron and solder was used to apply HT solder without oxidizing the metal.

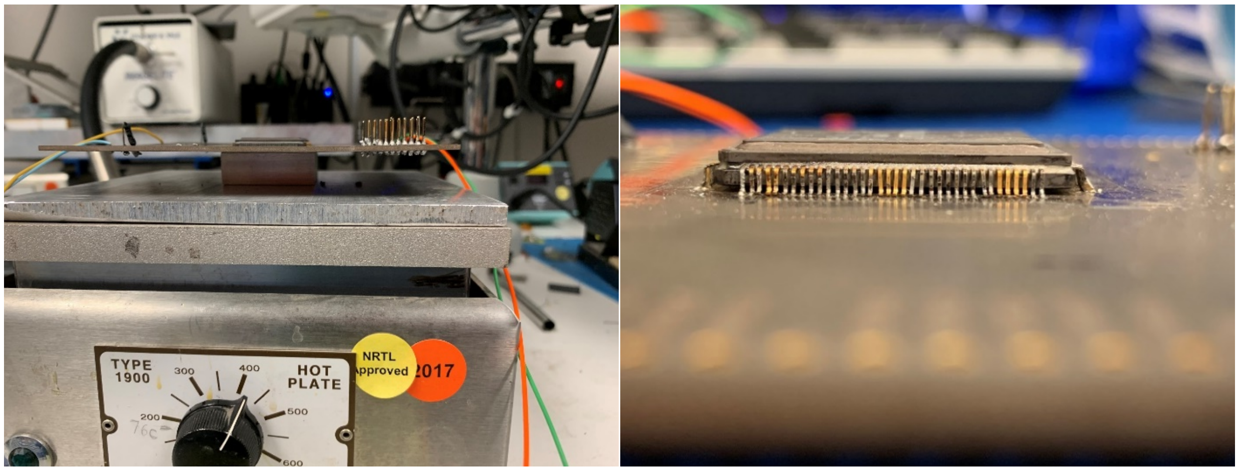


Figure 32. (Left) RC10001 package set on a hot plate with a pedestal underneath to focus the heat on the IC pins. (Right) RC10001 soldered to the PCB with HT solder only on the necessary pins.

The third package recycled the RC10001 from the second package but used a new PCB. The set was placed in the oven at 100–110°C for 41 hours to ensure there was no moisture in the board. Again, the metal weight was placed on top of the IC. Temperatures were increased slowly to obtain as much data as possible. The temperature was increased to 200˚C and the package sat for 117 hours with no issues. Next, the temperature increased to 220–234˚C and the package operated for 157 hours until failure. Unfortunately, the power lines shorted again before reaching 300˚C. Notably, the recycled microcontroller was exposed to 300˚C in an off state for 19 hours; however, this does not indicate the chip would function at 300˚C. Considering the results obtained from the SRAM chip, which uses the same transistor technology, the IC should function at 300˚C. During the three experiments, the current draw increased approximately 3–4 mA at elevated temperatures.

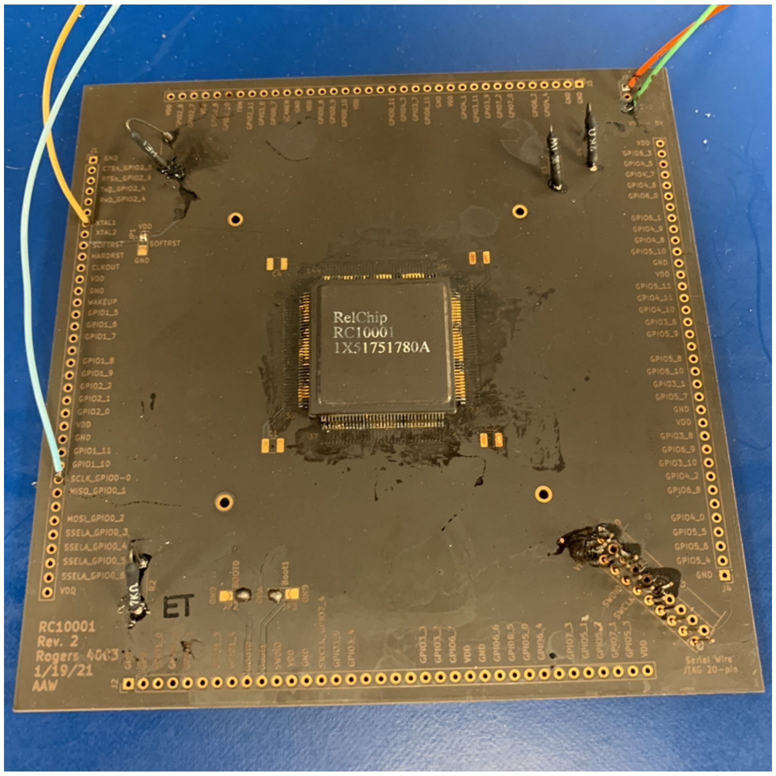


Figure 33. RC10001 and packaging exposed to a max of 234˚C for 157 hours.

2.4.4. Evaluating SM320F28335-HT

As noted previously, the software for the SM320F28335-HT was the most complex among the devices evaluated under in this study. That said, the packaging was the easiest to deal with largely because the microcontroller can only operate up to 210˚C and is a thru hole device. These factors made the packaging significantly more reliable compared to operating an SMD device at 300˚C. The evaluated data link is shown in Figure 34.

The setup used to evaluate the data link include the following: three external power supplies used to power the data link, including 1.9 V, 3.3 V and 12 V sources; a PicoScope oscilloscope used to receive the ODFM+BPSK signal from the data link output; and a 1524 m (5000 ft) single conductor wireline placed between the output of the data link and the input of the oscilloscope. The wireline was placed outside the oven because of its size. An on-board crystal oscillator was initially used to clock the microcontroller and PicoScope oscilloscope software was used to save the data. Finally, MATLAB was used in post-processing to display the messages and create a constellation plot.

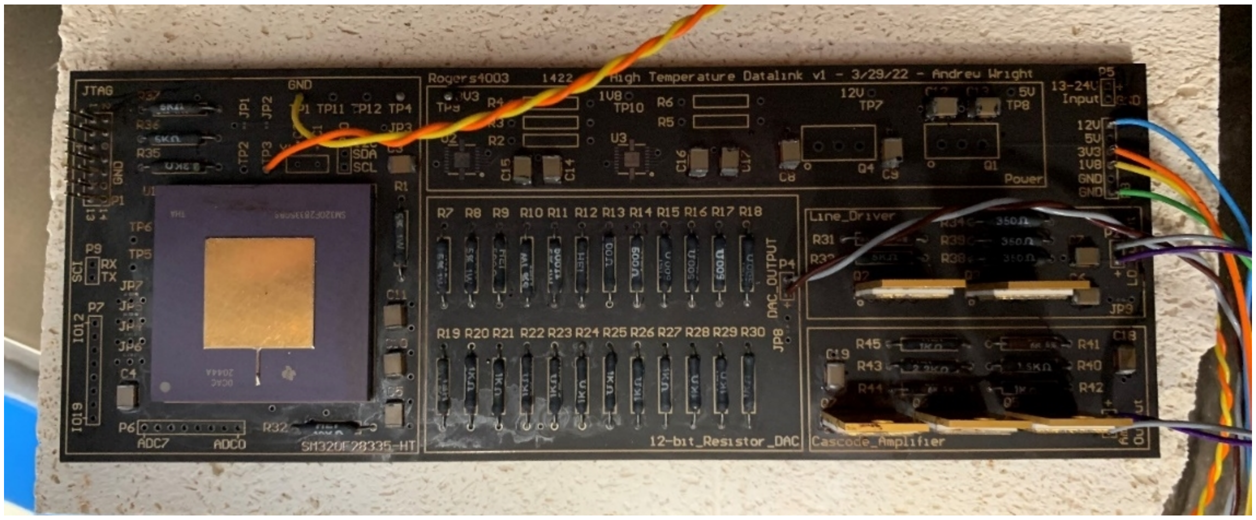


Figure 34. The data link after operating at 210˚C for 288 hours.

For the first elevated temperature test, the microcontroller current draw was 221 mA, 200 mA, and 75 mA for 1.9 V, 3.3 V, and 12 V, respectively. As a precaution, the board was heated to 100˚C for 24 hours to bake-out moisture. Following this, the temperature was increased but stopped at 110˚C when the transmitted messages became corrupted. After debugging the clock and software, we determined the error occurred because the microcontroller operated at 150 MHz. The code written for the microcontroller was built off the library for a device that operates at lower temperatures. According to the data sheet, the IC can operate at 150 MHz but only below 125˚C. It needs to use a derated clock of 100 MHz when operating at above 200˚C. The IC has a built-in clock multiplier which takes the external low clock signal and multiplies it to a higher frequency. A 30 MHz HT crystal was purchased for the board to match the low temperature sister board frequency. With that, the multiplier was adjusted in the code to get close to a 100 MHz internal clock; however, an exact 100 MHz could not be achieved due to the multiplier's limited settings. After this adjustment, the operating temperature of the board did not improve much. Thus, the crystal was removed and an external clock from a DAQ was used in its place to produce a 20 MHz signal that enabled an exact 100 MHz internal clock. With this correction, the data link could send messages up to 170˚C. We observed current draw of 176 mA, 185 mA, and 80 mA, for 1.9 V, 3.3 V, and 12 V, respectively, after updating the code and using the external clock. The next issue occurred because of the cascode amplifier and line driver. As the temperature increased the amplifier and line driver attenuated the signal prematurely. The amplifier circuit will need to be redesigned to compensate for the elevated temperature. In the next data link design, Honeywell's opamp will be used instead, considering it is most likely designed to compensate for elevated temperatures. To continue the study of the microcontroller, the amplifier, line driver, and 1524 m (5000 ft) wireline were bypassed, and the signal was received directly from the output of the DAC. With the bypass, the messages were received to 210˚C. The chip was then tested at 210˚C for 288 hours with no issues.

Figure 35 shows the constellation diagram for the data link. The left image is the constellation from the output of the wireline at 170˚C and the right image is the constellation from the output of DAC at 210˚C. In the left image, the points are further spread relative to room temperature operation because of thermal noise. As the temperature increased, noise was generated on all conductive elements. Unfortunately, the noise cannot be prevented, but various techniques can be used to confirm messages from the data link are received correctly. Results from this effort can be used to significantly improve the performance of the data link including data rates, operating temperature, and noise performance.

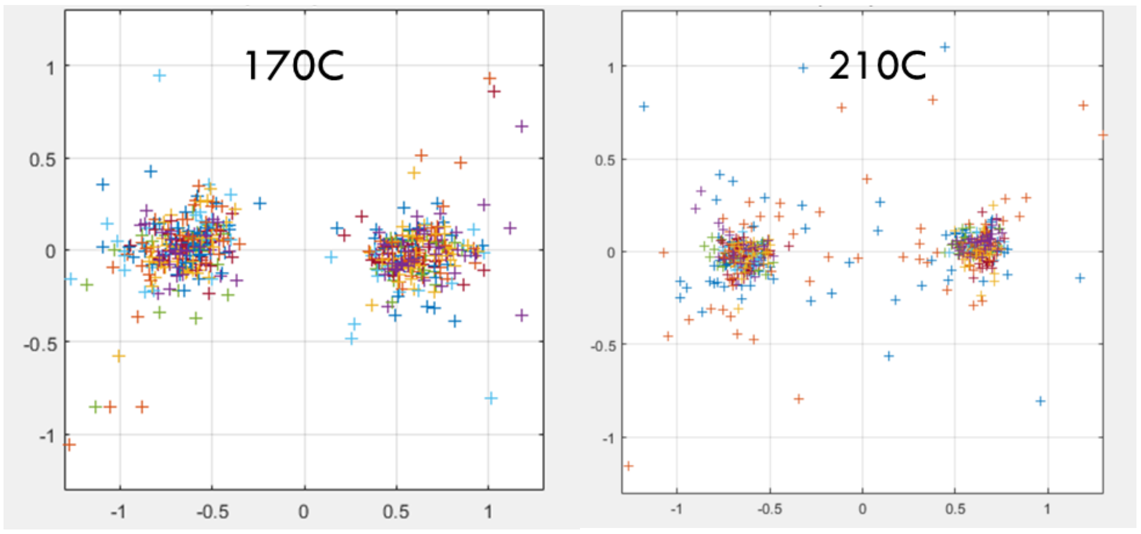


Figure 35. Constellation plot for the full data link and 1524 m (5000 ft) wireline (Left) and plot for the data link when bypassing the amplifier, line driver, and 1524 m (5000 ft) wireline (Right).

3. CONCLUSION

We evaluated five microcontrollers, one memory IC, and a data link designed to operate at or above 210°C under this effort sponsored by Department of Energy Geothermal Technology Office. Figure 5 shows the five microcontrollers we evaluated, and we recommend the three highlighted ones for use in geothermal applications. These recommendations are based on the ability to program the device and/or good performance results. SM320F2812-HT and SM470R1B1M-HT could not be programmed and were not evaluated under this effort. However, considering an external company's success programming the SM470R1B1M-HT and its small form factor, we decided to recommend the device for geothermal applications. SM320F28335-HT is highly recommended as it is easily programmable with Code Composer or MATLAB. Considering the device is a thru hole device, it makes packaging highly reliable. Despite its temperature capabilities, the HT83C51 was not recommended because of its limited capabilities and programming difficulties for new users. RC10001 is highly recommended for geothermal applications assuming the device remains available. As of 2022, RelChip is being acquired by another company, leaving the future of the RC10001 uncertain. However, if the IC continues to be manufactured, it is the only device rated to operate at 300°C temperatures, making it unique compared to other options.

Table 5. Recommendations for MCUs Evaluated Under this Effort.

Device	Comments	Recommendations
SM320F2812-HT	Outdated microcontroller. Was not able to detect device with TI software, thus could not program device.	Not Recommended
SM470R1B1M-HT	Outdated microcontroller. Successfully detected the device with KEIL software but could not be programmed. IAR Systems software maybe able to interface with it.	External company claimed to successfully program device
SM320F28335-HT	Successfully programmed with both TI Code Composer and MATLAB.	Recommended for use
HT83C51	Outdated microcontroller with minimal features. Successfully programmed with custom programmer.	Not recommended
RC10001	Successfully programmed with Keil. Issues with device delaminated from PCB. Owner is currently in the process of selling the company, device maybe discontinued.	Recommended if component is still available

SNL designed and evaluated a data link with the SM320F28335-HT microcontroller. It operated fully through a 1524 m (5000 ft) single conductor wireline up to 170°C. The full data link did not operate up to 210°C due to limitations with the amplifier stage. After removing the amplifier and line driver stages from the data link and directly connecting the oscilloscope to the output of the DAC, the data link transmitted messages up to 210°C. Using the OFDM+BPSK communication technique, it yielded 30 kbps data rates at elevated temperatures. For future testing, the PCB was designed to accommodate voltage regulation and access pins for the analog-to-digital converter (ADC), allowing for power transfer through the line and the ability to receive data.

3.1. Future Efforts

A major problem in this research was packaging failures, which severely limited testing on the RC10001 microcontroller. The team observed deformation of the PCB, degradation of the solder, CTE mismatch, delamination of the ICs, and shorting of internal traces. To improve future reliability of the packaged device, the Rogers PCB will be replaced with a ceramic PCB, and we will use either gold/tin solder bonding or wire bonding as well as an HT conformal coating. These will hopefully resolve the various issues observed during testing. Ceramic can survive much higher temperatures than 300˚C. Gold/tin solder would be ideal for the gold pins on the IC as the solder joint would not degrade. An alternative method would be wire bonding, which fuses gold wire to the pins and to the PCB pads. The wire is capable of operating at temperatures above 300˚C and would allow mechanical movement between the IC and PCB (CTE mismatch) at elevated temperatures. Finally, the conformal coating, a layer of cured liquid epoxy covering the IC and PCB, could improve the strength between the IC and PCB as well as prevent oxidation to the metals. This packaging improvement technique could significantly improve the reliability of the logging tool electronics, which will be exposed to high temperatures, mechanical shock, and vibration.

Future improvements to the data link can be conducted to improve data rates, operating temperatures, signal-to-noise ratio (SNR), error rates, and receiving data. Operating temperatures can be accomplished by replacing the SM320F28335-HT with the RC10001. Data can be received using an ADC, which would enable signals to be received from the surface. Data rates can significantly be increased by increasing the constellation point, which can be done by incorporating QAM instead of BPSK. To use QAM, the signal integrity will need to be improved. Signal integrity can be improved by channel coding, error correction code, constant signal propagation profile, increasing amplifier gain, and using a pulse width modulation (PWM) DAC. Channel coding would include coding steps that work together to mitigate the effects of channel distortions. These steps may include interleaving the data followed by applying error correction code. Error correction code, such as the Bose-Chaudhuri-Hocquenghem technique, can be incorporated to improve the SNR of the system. The profile of wireline drops above 13 kHz which can be compensated for by an amplifier. With a constant propagation profile, the amplitude modulation can be constant to the maximum desired operating bandwidth. Increasing the gain performance of the amplifier could further push the signal above the noise floor, mitigating noise degradation to the signal. Finally, the resistor array DAC could be replaced with a PWM DAC. This method would result in less resistor inaccuracies. The current resistors did not divide the rail voltage accurately and evenly, which injected some noise into the signal.

These improvements to the data link would allow for reliable real-time data collection up to 300˚C. This technology can be used in various applications such as subsurface array system capturing tracer data or acoustics.

REFERENCES

[1] Burroughs, C. “Hot Research at Sandia may Make Producing Electricity from Geothermal Energy More Cost Competitive.” Sandia National Laboratories News, Albuquerque, NM (1998).

[2] Demeus, L., Delatte, P., Dessard, V., Adriaensen, S., Viviani, A., Renaux, C., and Flandre, D. “The Art of High Temperature FD-SOI CMOS.” IEEE (1999).

[3] Henfling, J., and Norman, R. “Dewarless Logging Tool – 1st Generation.” SAND2000-1680, Sandia National Laboratories, Albuquerque, NM (2000).

[4] Normann, R., and Henfling, J. “Elimination of Heat-Shielding for Geothermal Tools Operating up to 300 Degrees Celsius.” Proceedings World Geothermal Congress, Kyushu – Tohoku, Japan (2000).

[5] Henfling, J., Normann, R. “High Temperature Downhole Reservoir Monitoring System.” Proceedings Twenty-Ninth Workshop on Geothermal Reservoir Engineering, Stanford University, California (2004).

[6] Rogers, J., Ohme, B., and Normann, R. “New Paradigm in Electronics Needed to Take the Heat of Deep Gas Drilling.” The American Oil & Gas Reporter (2005)

[7] Swenson, G., and Ohme, B. “HTMOS™: Affordable High Temperature Product Line.” Aerospace Honeywell.

[8] Reed, L. “A 250˚C ASIC Technology” HiTEN (2013)

[9] Normann, R., and Glowka, D. “Designing Logging Tools for Future High Entropy Geothermal Power.” GRC Transactions, Vol. 38 (2014).

[10] Zheng, W. et al. “A frequency domain scheme for high speed telemetry down hole wire line communication.” Int. Conf. Instrum. Meas. Comput. Commun. Control 1413–1417 (2015).

[11] Tran, T., Sun, W., Zeng, J. & Wiecek, B. “High-bitrate downhole telemetry system.” 2015 IEEE Int. Symp. Power Line Commun. Its Appl. ISPLC 2015 280–284 (2015).

[12] Liyanage, M. et al. “Evolution of Wireline Telemetry and its Impact on Formation Evaluation.” 22nd Form. Eval. Symp. Japan 22, (2016).

[13] Cheng, L., Jinfeng, C., Shangchun, F. & Jun, Y. “Analyzing the validity of a DFT-based improved acoustic OFDM transmission along rotating simulated drillstring.” Mech. Syst. Signal Process. 81, 447–460 (2016).

[14] Middlestead, R. W. “Digital Communications with Emphasis on Data Modems.” (Wiley, 2017).

[15] Cieslewski, G., and Cashion, A. “High Temperature Quadrature Amplitude Modulation of Orthogonal Frequency Division Multiplexing.” Volume 2017, Hiten (2017).

[16] Soares, M., Durham, W., and Behbahani, A. “A 0.15um SOI High Temperature ARM Microcontroller for Local Control Nodes: Status of the Next Step.” AIAA Propulsion and Energy Forum, Cincinnati, Ohio (2018).

[17] Wright, A., Cashion, A., and Tiong, F. “Evaluation of High Temperature Microcontrollers and Memory Chips for Geothermal Applications.” GRC2022 Conference, Reno, Nevada (2022).

[18] Wright, A., and Cashion, A. “High Temperature High Speed Data Transfer (Data Link).” GRC2022 Conference, Reno, Nevada (2022).

APPENDIX A. MAIN APPENDIX TITLE

A.1. HT83C51 Hello World Code

```
#include <8052.h>
#include <stdint.h>
#include <stdio.h>

#define TOGGLE_IO_0 (__xdata uint8_t*) 0x0100
#define TOGGLE_IO_5 (__xdata uint8_t*) 0x0200
#define MEMORY_MUX_CLEAR (__xdata uint8_t*) 0x0300
#define DATA_MUX_CLEAR (__xdata uint8_t*) 0x2400
#define DATA_MUX_SELECT (__xdata uint8_t*) 0x2300
#define ADC_123_CONVERT (__xdata uint8_t*) 0x2500
#define ADC_123_BYTE_SELECT (__xdata uint8_t*) 0x2600
#define ADC_123_MUX_SELECT (__xdata uint8_t*) 0x2700
#define ADC_123_MUX_CLEAR (__xdata uint8_t*) 0x2800
#define DATA_BUS_SELECT (__xdata uint8_t*) 0x2900
#define ADC_1_CS (__xdata uint8_t*) 0x2A00
#define ADC_2_CS (__xdata uint8_t*) 0x2B00
#define ADC_3_CS (__xdata uint8_t*) 0x2C00
#define TOGGLE_IO_1 (__xdata uint8_t*) 0x3000
#define SPINNER_BYTE_SELECT (__xdata uint8_t*) 0x3200
#define SPINNER_CLEAR_START (__xdata uint8_t*) 0x3300
#define ASIC_RESET (__xdata uint8_t*) 0x3C00

#define SAMPLE_NUM 40

#define CLK_5MHZ

#ifdef CLK_5MHZ
        #define COUNT_20US 0xFFF6

#else
        #define COUNT_20US 0xFFEC
#endif


uint8_t toggleasic(__xdata uint8_t* address);
void toggleasicx(__xdata uint8_t* address, uint8_t num);
void setup();
void putchar(char c);
char getchar();
void delay(uint16_t count);
void printf_pico(__code char* str);
void printf_hex(uint8_t num);
uint8_t nibble2ascii(uint8_t num);
```

```
uint16_t adc_read(uint8_t channel);
void spinner_read();
void send_packetASCII(uint32_t frame_count);
void send_packet(uint32_t frame_count);

//Global Variables
volatile __bit TI_safe = 0;
volatile __xdata uint8_t spinner_data[3];
volatile __xdata uint8_t spinner_isr_data[3];
volatile __xdata uint32_t adc_filt_data[7];
volatile __xdata uint16_t adc_data_1[7];
volatile __xdata uint16_t adc_data_2[7];
volatile __xdata uint32_t cc_filt_data;
volatile __xdata uint16_t cc_data_1[SAMPLE_NUM];
volatile __xdata uint16_t cc_data_2[SAMPLE_NUM];
uint8_t sec_counter = 6;
volatile __bit osef = 0;
volatile __bit data_ready = 0;
volatile uint8_t sample_count = 0;
volatile __bit buffer_flag = 0;


void main()
{
	uint32_t frame_count = 0;
	//Code
	EA = 0;
	setup();

	printf_pico("Welcome to the Monitor Program\r\n");
	TR2 = 1;
	while(1)
	{
		if(data_ready)
		{
			//P1_2 = 1;
			//data_ready = 0;
			//send_packetASCII(frame_count);
			//frame_count++;
			//P1_2 = 0;

		printf_pico("Hello World\r\n");
		}
	}
}

//It seems that the HT83C51 likes to hang when pooling on the TI flag directly
//This ISR bypasses this problem
```

```
void serial_isr(void) __interrupt(4)
{
	if(TI==1)
	{
		TI_safe = 1;
		TI = 0;
	}
	if(RI==1)	//Return to the bootloader (fails to reprogram correctly after that?)
	{
		RI = 0;
		/*if(SBUF == 0x03)
		{
			__asm
			LJMP 0x0000
			__endasm;
		}*/
	}
}

//Putchar is not allowed in the ISR, it will hang!
void timer2_isr(void) __interrupt(5)
{
	volatile __xdata uint16_t* cc_data;
	volatile __xdata uint16_t* adc_data;

	TF2 = 0;	//Reset the timer flag
	P1_3 = 1;	//Debug

	//Select correct buffer do put data into
	if(!buffer_flag)
	{
		cc_data = cc_data_1;
		adc_data = adc_data_1;
	}
	else
	{
		cc_data = cc_data_2;
		adc_data = adc_data_2;
	}

	//Sample the data and store
	cc_data[sample_count] = adc_read(9); //colar Counter
	adc_filt_data[0] += adc_read(10);//Other Sensors
	adc_filt_data[1] += adc_read(11);
	adc_filt_data[2] += adc_read(12);
	adc_filt_data[3] += adc_read(13);
	adc_filt_data[4] += adc_read(14);
	adc_filt_data[5] += adc_read(15);
```

```
        adc_filt_data[6] += adc_read(16);
        sample_count++;

        //Sample spinner every 6s; base on the one second signal from ASIC
        if(P1_5 != osef)
        {
                osef = !osef;
                sec_counter--;
        }

        if(sec_counter <= 0)
        {
                sec_counter = 6;
                spinner_read();
        }

        //if SAMPLE_NUM reached average the slow sensors and set data ready flag
        //Copy the spinner_isr_data into spinner data for the output purposes
        if(sample_count>=SAMPLE_NUM)
        {
                uint8_t i = 0;
                for(i = 0; i<7; i++ )
                {
                        adc_data[i] = adc_filt_data[i]/SAMPLE_NUM;
                        adc_filt_data[i] = 0;
                }

                spinner_data[0] = spinner_isr_data[0];
                spinner_data[1] = spinner_isr_data[1];
                spinner_data[2] = spinner_isr_data[2];

                sample_count = 0;
                data_ready = 1;
                buffer_flag = !buffer_flag;
        }


        P1_3 = 0;
}

//The counter counts up so the value set must be 0XFFFF-numbercycles to count
void delay(uint16_t count)      //Using timer 0
{
        TL0 = (uint8_t) count;
        TH0 = (uint8_t) (count >> 8);
        TR0 = 1;
        while(!TF0);
        TR0 = 0;
```

```
	TF0 = 0;
}

uint16_t adc_read(uint8_t channel)
{

	uint16_t value = 0;
	P1_1 = 1;
	toggleasic(ADC_123_MUX_CLEAR);//Clear analog mux
	toggleasicx(ADC_123_MUX_SELECT, channel-1);	//Select output of analog mux
	toggleasic(ADC_123_CONVERT); //The original assembly code converts twice?
	delay(COUNT_20US);
	toggleasic(ADC_123_CONVERT); //Convert
	delay(COUNT_20US);//Wait 20us
	//Data out of the ADC.  1st byte bits 11-7, 2nd byte bits 3-0 (they are located in the uper
nibble of the byte)
	value = toggleasic(ADC_1_CS) << 4;
	toggleasic(ADC_123_BYTE_SELECT);
	value |= toggleasic(ADC_1_CS) >> 4;
	toggleasic(ADC_123_BYTE_SELECT);
	P1_1 = 0;
	return value;
}

//Read the spinner values.  Assumes the data is  ready.  Can be read every 6s.  Returns 2*spin
rate/s.
void spinner_read()
{
	toggleasic(DATA_MUX_CLEAR);
	toggleasicx(DATA_MUX_SELECT, 11);
	//Read 1st byte
	spinner_isr_data[0] = toggleasic(DATA_BUS_SELECT);
	toggleasic(SPINNER_BYTE_SELECT);
	//Read 2nd byte
	spinner_isr_data[1] = toggleasic(DATA_BUS_SELECT);
	toggleasic(SPINNER_BYTE_SELECT);
	//Read 3rd byte
	spinner_isr_data[2] = toggleasic(DATA_BUS_SELECT);
	toggleasic(SPINNER_BYTE_SELECT);
	//Reset Spin Count
	toggleasic(SPINNER_CLEAR_START);

}

void send_packet(uint32_t frame_count)
{
	__xdata uint16_t* cc_data;
	__xdata uint16_t* adc_data;
```

```
        uint8_t i;

        if(buffer_flag)
        {
                cc_data = cc_data_1;
                adc_data = adc_data_1;
        }
        else
        {
                cc_data = cc_data_2;
                adc_data = adc_data_2;
        }
        //Send Header 4bytes
        putchar(0xAA);
        putchar(0x55);
        putchar(0xAA);
        putchar(0x55);

        //Send frame_count 4 bytes
        putchar(frame_count >> 24);
        putchar(frame_count >> 16);
        putchar(frame_count >> 8);
        putchar(frame_count);

        //Send spiner data 3 bytes
        putchar(spinner_data[0]);
        putchar(spinner_data[1]);
        putchar(spinner_data[2]);

        //Send ADC data 7*2 = 14 bytes
        for(i=0; i<7; i++)
        {
                putchar(adc_data[i]>>8);
                putchar(adc_data[i]);
        }
        //Send Colar Counter readings 2*SAMPLE_NUM
        for(i=0; i<SAMPLE_NUM; i++)
        {
                putchar(cc_data[i]>>8);
                putchar(cc_data[i]);
        }
        //Send data_ready flag to make sure we are not overflowing 1byte
                putchar(data_ready);

        //total bytes sent = 154


}
```

```
void send_packetASCII(uint32_t frame_count)
{
	__xdata uint16_t* cc_data;
	__xdata uint16_t* adc_data;
	uint8_t i,j;

	if(buffer_flag)
	{
		cc_data = cc_data_1;
		adc_data = adc_data_1;
	}
	else
	{
		cc_data = cc_data_2;
		adc_data = adc_data_2;
	}
	//Send Header 1bytes
	putchar('+');

	//Send frame_count 6 bytes
	printf_hex(frame_count >> 16);
	printf_hex(frame_count >> 8);
	printf_hex(frame_count);

	//Send spiner data 6 bytes
	printf_hex(spinner_data[0]);
	printf_hex(spinner_data[1]);
	printf_hex(spinner_data[2]);

	//This loop is setup for SAMPLE_NUM = 40!!
	//Send ADC data 7*(5+1)*4 = 168 bytes
	for(i=0; i<7; i++)
	{
		for(j=0; j<5; j++)
		{
			printf_hex(cc_data[5*i+j]>>8);
			printf_hex(cc_data[5*i+j]);
		}
		printf_hex(adc_data[i]>>8);
		printf_hex(adc_data[i]);

	}
	//send last CC data set 4*5  = 20 bytes
	for(j=0; j<5; j++)
	{
		printf_hex(cc_data[5*i+j]>>8);
		printf_hex(cc_data[5*i+j]);
	}
	//Send data_ready flag to make sure we are not overflowing 3byte
		printf_hex(data_ready);
		putchar('\r');
		putchar('\n');

	//total bytes sent = 204
}

//if this it changed whole code will stop working!
uint8_t toggleasic(__xdata uint8_t* address)
{
	uint8_t temp = 0;
	P1_7 = 1;
	temp = *address;
	P1_7 = 0;
	return temp;
}

void toggleasicx(__xdata uint8_t* address, uint8_t num)
{
	while(num>0)
	{
		toggleasic(address);
		num--;
	}
}

void setup()
{
	IE = 0;//Disable all interrupts


	P1_7 = 0;
	P1_5 = 1; //Set for input
	P1_6 = 1; //Set for input

	toggleasic(ASIC_RESET);
	toggleasic(DATA_MUX_CLEAR);
	toggleasic(MEMORY_MUX_CLEAR);
	toggleasic(TOGGLE_IO_1); //Power to the Reed switches in SC
	toggleasic(SPINNER_CLEAR_START);
	//Serial Port Setup and Timer 0,1


	PCON |= 0x80;		//Set SMOD bit in PCON for 5.5296 Mhz clock

	TH1 = 0xF4; //Set baud rate to 2400
	TL1 = 0x00;
	TMOD = 0x21;

	TCON = 0x40;
	SCON = 0x52;
	TR1 = 1; //Enable the timer1 for serial port

	RI = 0;
	TI = 0;

	//Setup interrupts
	IP = 0; //Reset interrupt priorities
	ES = 1; //Enable serial interrupt
	TI_safe = 1; //Enable local ready to send flag
	PS = 1; //Elevate priroirity of serial interrupt for testing

	//Timer 2 determines sampling frequency.

	ET2 = 1; //Enable timer2 interrupt

	TH2 = 0xC3;   //for the 5.5296 Mhz clock and ASCII aoutput 30.006hz sampling freq
	TL2 = 0xFF;   //MATLAB: dec2hex(0xFFFF-round((5.5296e6/12)/30))
	RCAP2H = 0xC3;
	RCAP2L = 0xFF;

	EA = 1; //Enable interrups
}

void putchar(char c) //This function is required for the printf_tiny() to work
{
	while(TI_safe==0); //Wait for the TI flag to be set
	TI_safe = 0; //Clear the TI flag
	SBUF = c;
	return;
}
char getchar() //This function is required for the printf_tiny() to work
{
	char temp;
	while(RI==0); //wait for the RI flag to be set
	temp = SBUF;
	RI = 0;
	return temp;
}
void sleep(uint8_t count)
{
	uint8_t i;
	for(i=0; i<count; i++);
}
```

```
void printf_pico(__code char* str)
{
	uint8_t i;
	for(i=0; str[i]!=0; i++)
	{
		putchar(str[i]);
	}
}

void printf_hex(uint8_t num)
{
	putchar(nibble2ascii((num >> 4) & 0x0F));
	putchar(nibble2ascii(num & 0x0F));
}

uint8_t nibble2ascii(uint8_t num)
{
	if(num < 10)
	{
		return num + 48;
	}
	else if (num <= 16)
	{
		return num + 55;
	}
	else
	{
		return 'X';
	}
}
```

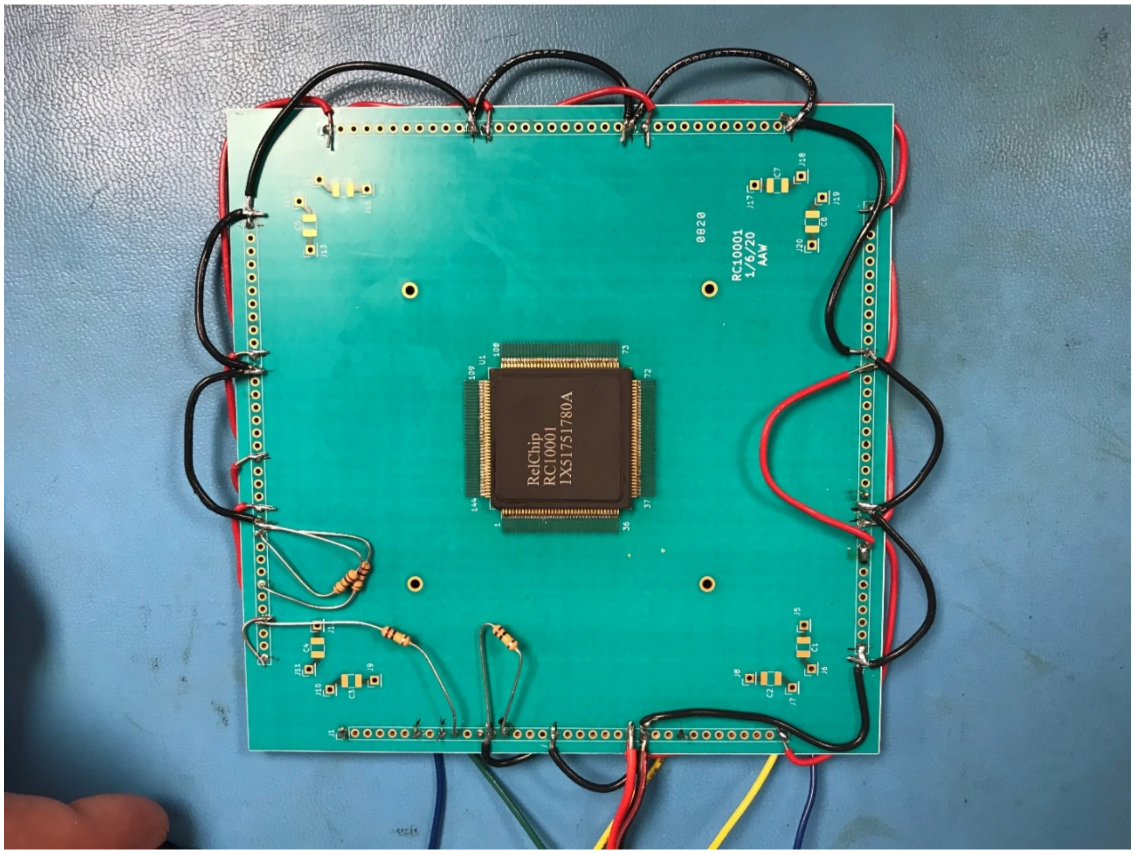
A.2. RC10001 Pin Connection and Keil Software Setup

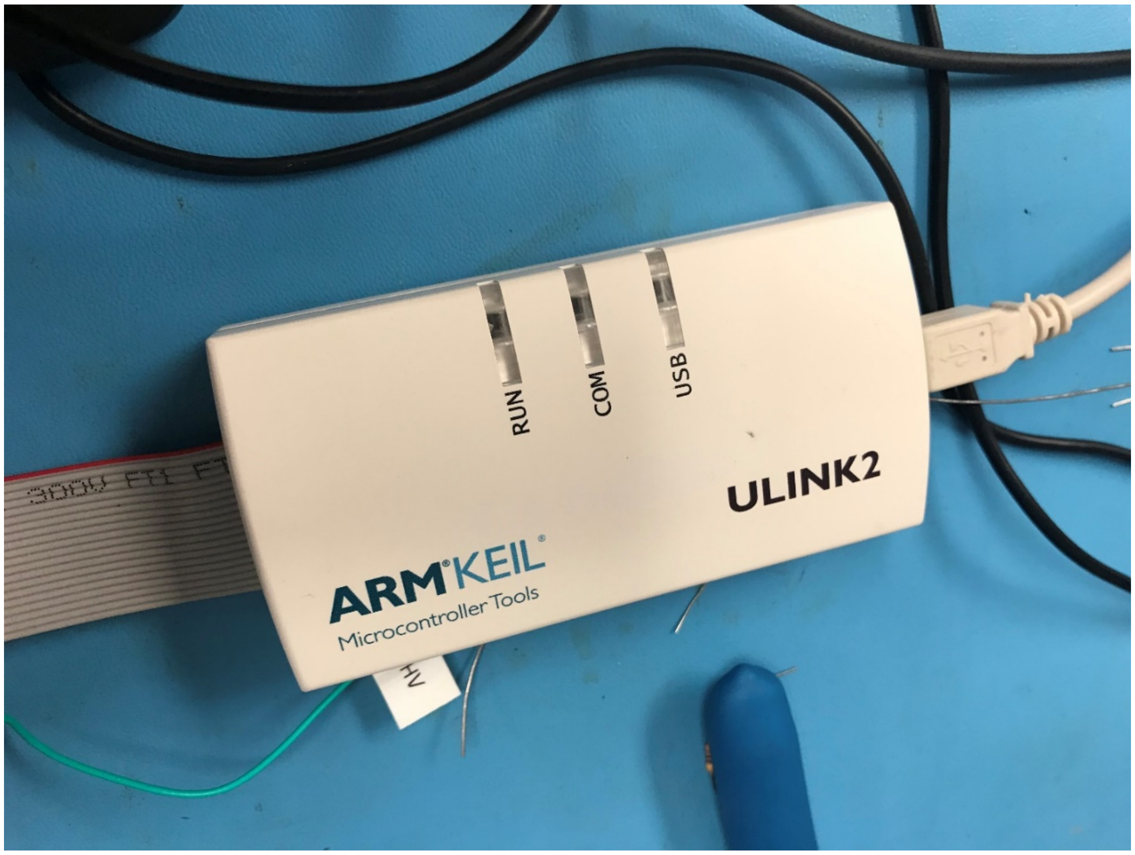
To program the Relchip on its own board, away from the development board:

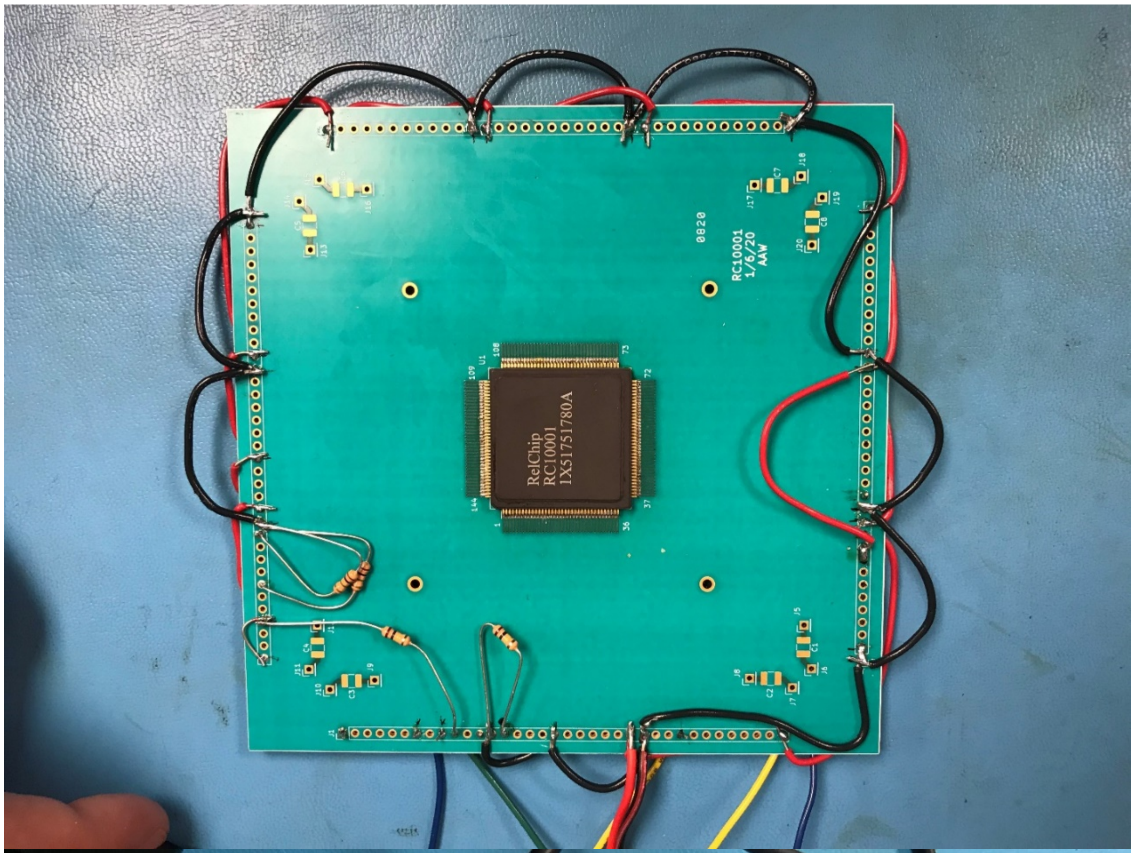
- Apply clock to XTAL1 pin. XTAL2 is float. The clock is supposed to be CMOS input but we found a sine wave to be more stable to avoid ringing effects over long cables. The sine wave goes from minimum 0V to maximum 5V. Make sure the waveform generator is in "High Z" mode and not in 50 Ohm. We did 3.6 MHz.
- The two boot select pins (45 and 46) need to be held high at 5V in order to set the chip to be programmed from the SWD interface. You can just pull them high and leave them that way if you plan to just use SWD. The development board pulls them high using 100 kΩ resistors. We just soldered a wire to hold them high.
- SCANEN (Pin 134) and TRM (Pin 138) should be pulled low with 1MΩ resistors.
- HARDRST (Pin 10) is pulled high with a 100 kΩ resistor.

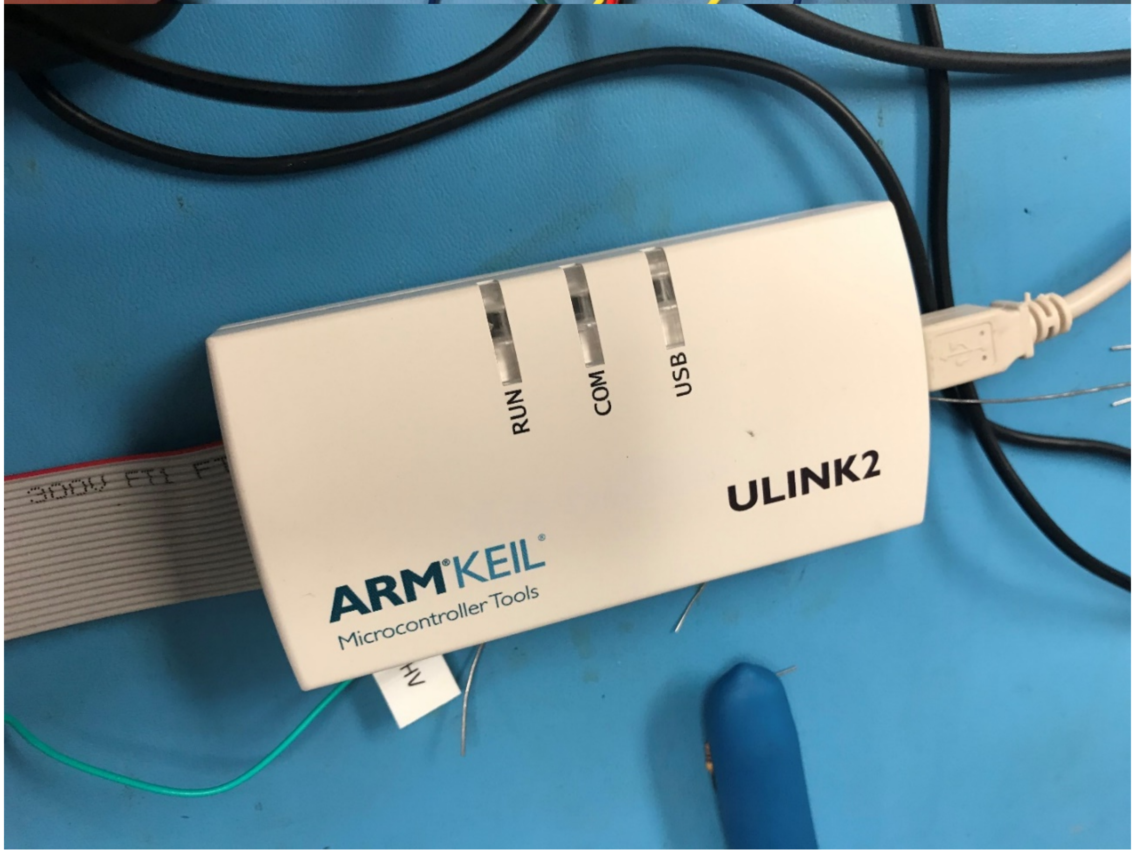
- WAKEUP (Pin 14) is pulled low with a 100 kΩ resistor.
- SOFT Reset is pulled high.
- The programmer used is the ARM KEIL ULINK2 device. The wires required are VDD, GND, SWCLK, and SWDIO. The development kit uses the ULINK-ME device but it is not practical for developing your own board because the ULINK-ME device is a 3.3V devices and the RC10001 is a 5V device. The development board has a bidirectional level shifter circuit to convert 3.3 V to 5 V and vice-versa. The UNLINK2 can be used without any level shift.
- GPIO_0 blinks using the BLINKY project.
- There is no flash on this chip so you must turn off the flash programming options in the target options menus in uvision (Keil software). The chip actually gets programmed by starting the debugger. Debug > Start/Stop Debug Session. Then click Run code. Once you run the code in the debugger, you can unplug the ULINK2 and the code will keep running on the chip.
- We used Keil uvision5 (MDK-Arm) to program the chip using the 3rd party uvision software pack for the Relchip part provided by RelChip.
- https://www.keil.com/download/product/
- Require Keil MDK v4 Legacy Support Software Pack, it contains the RC10001 within the library. The chip is a Cortex-M device.
- https://www2.keil.com/mdk5/legacy/

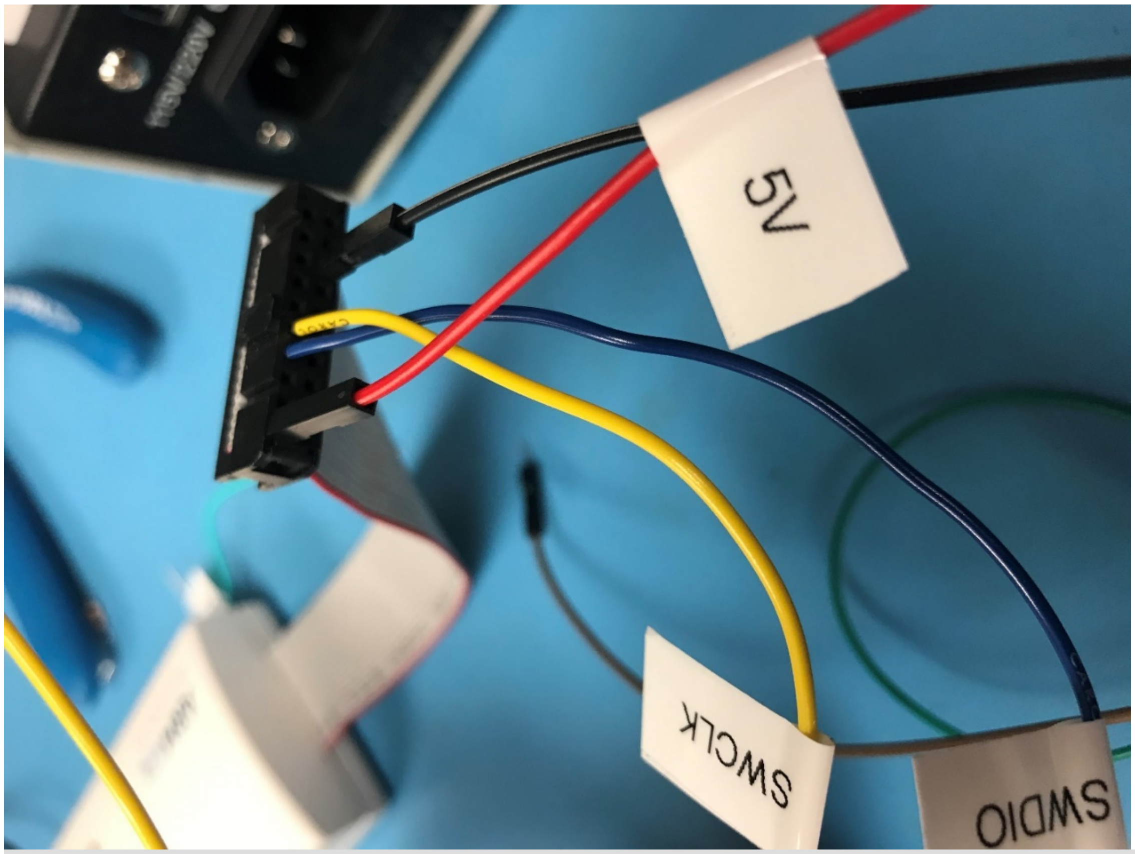
Target Options Settings:

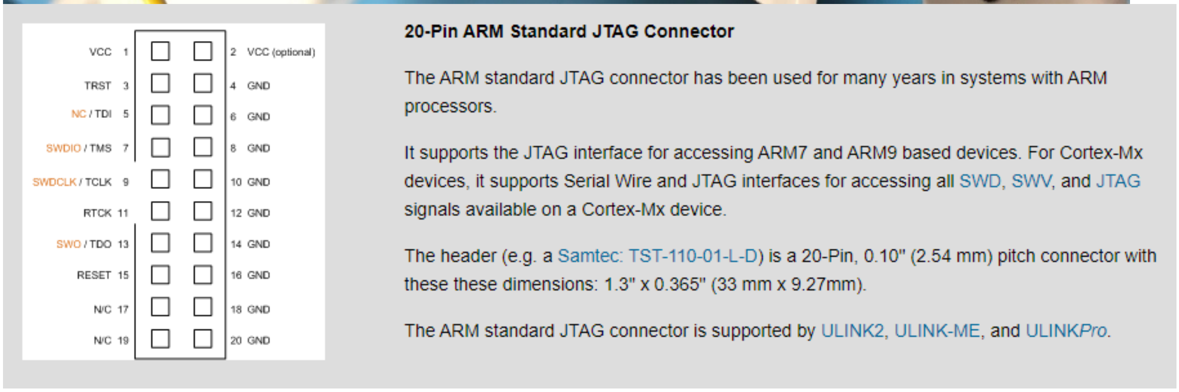

RelChip
RC10001
1X5175178OA
RC10001
1/6/20
AAW


RUN
COM
USB
ULINK2
ARM KEIL
Microcontroller Tools


RelChip
RC10001
1X51751780A
RC10001
1/6/20
AAW
0820


RUN
COM
USB
ULINK2
ARM KEIL
Microcontroller Tools





20-Pin ARM Standard JTAG Connector

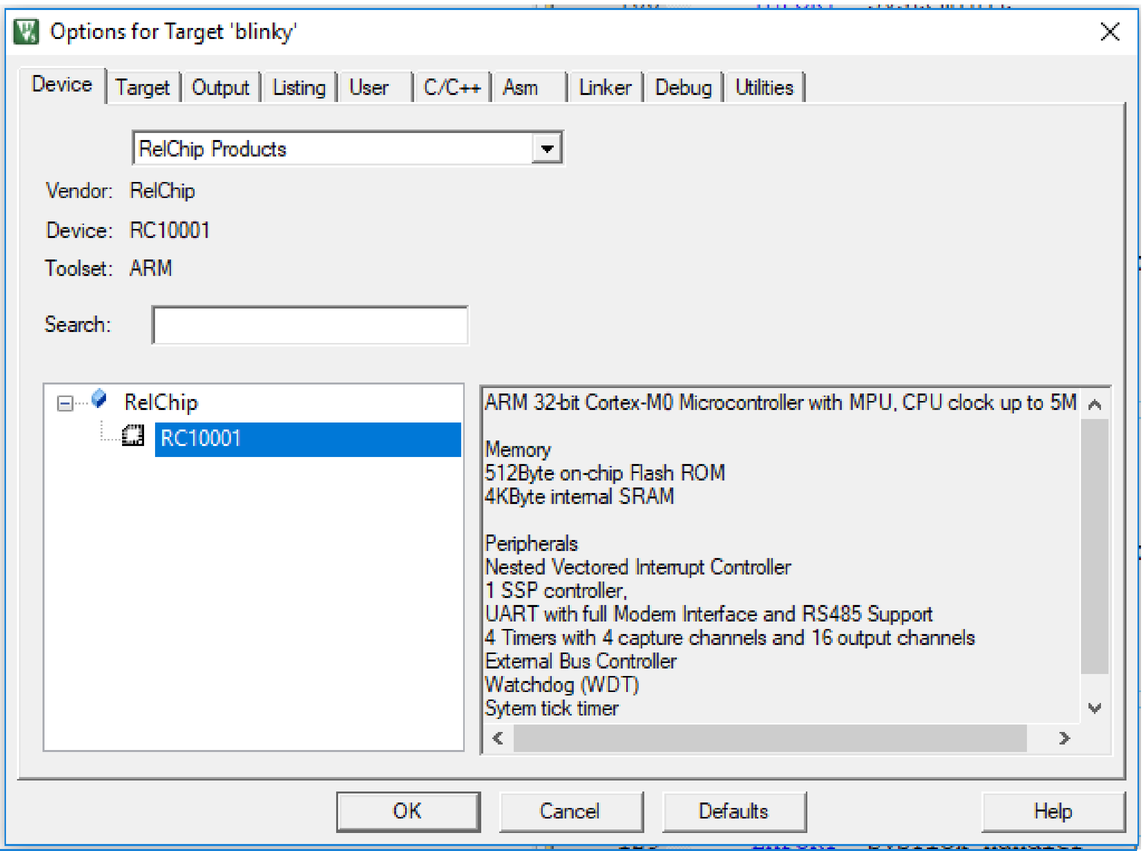
The ARM standard JTAG connector has been used for many years in systems with ARM processors.

It supports the JTAG interface for accessing ARM7 and ARM9 based devices. For Cortex-Mx devices, it supports Serial Wire and JTAG interfaces for accessing all SWD, SWV, and JTAG signals available on a Cortex-Mx device.

The header (e.g. a Samtec: TST-110-01-L-D) is a 20-Pin, 0.10" (2.54 mm) pitch connector with these these dimensions: 1.3" x 0.365" (33 mm x 9.27mm).

The ARM standard JTAG connector is supported by ULINK2, ULINK-ME, and ULINK*Pro*.

Target Options Settings:


Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

RelChip Products

Vendor: RelChip
Device: RC10001
Toolset: ARM

Search:

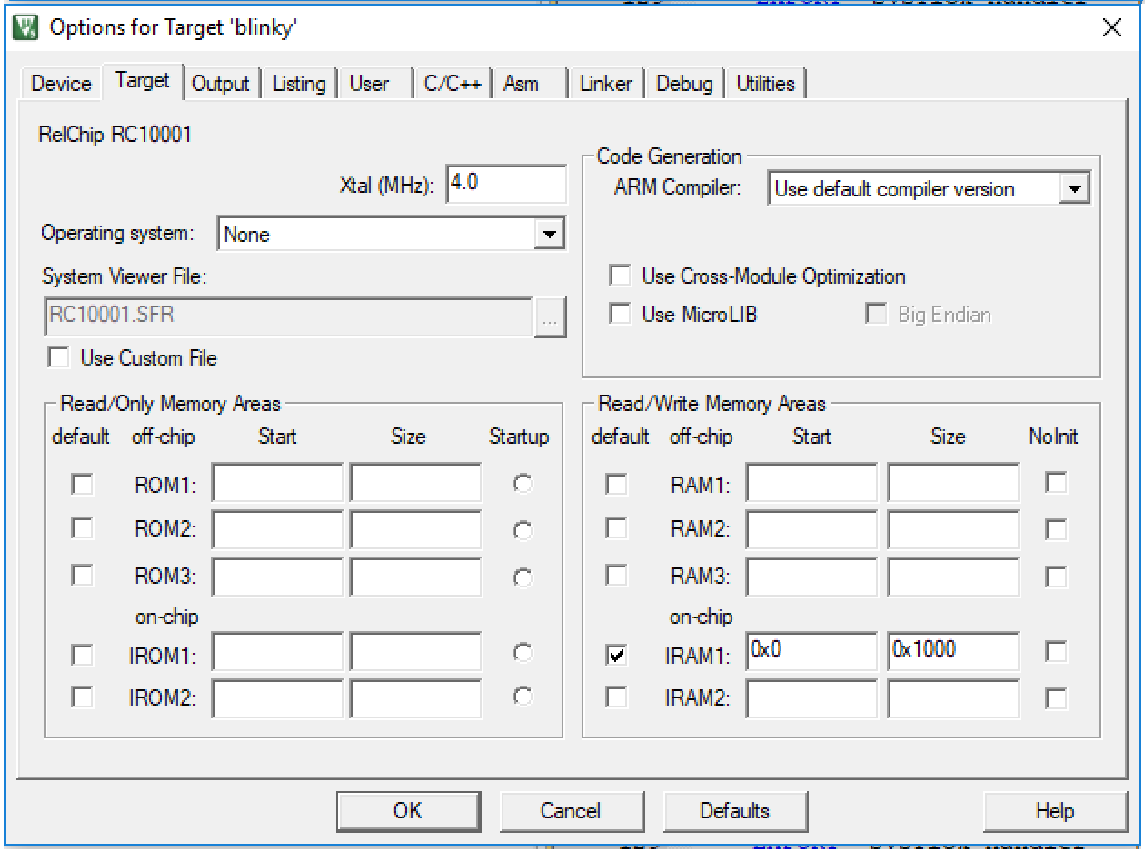
RelChip
RC10001

ARM 32-bit Cortex-M0 Microcontroller with MPU, CPU clock up to 5M

Memory
512Byte on-chip Flash ROM
4KByte internal SRAM

Peripherals
Nested Vectored Interrupt Controller
1 SSP controller,
UART with full Modem Interface and RS485 Support
4 Timers with 4 capture channels and 16 output channels
External Bus Controller
Watchdog (WDT)
Sytem tick timer

OK | Cancel | Defaults | Help


Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

RelChip RC10001

Xtal (MHz): 4.0

Operating system: None

System Viewer File: RC10001.SFR

☐ Use Custom File

Code Generation
ARM Compiler: Use default compiler version

☐ Use Cross-Module Optimization
☐ Use MicroLIB ☐ Big Endian

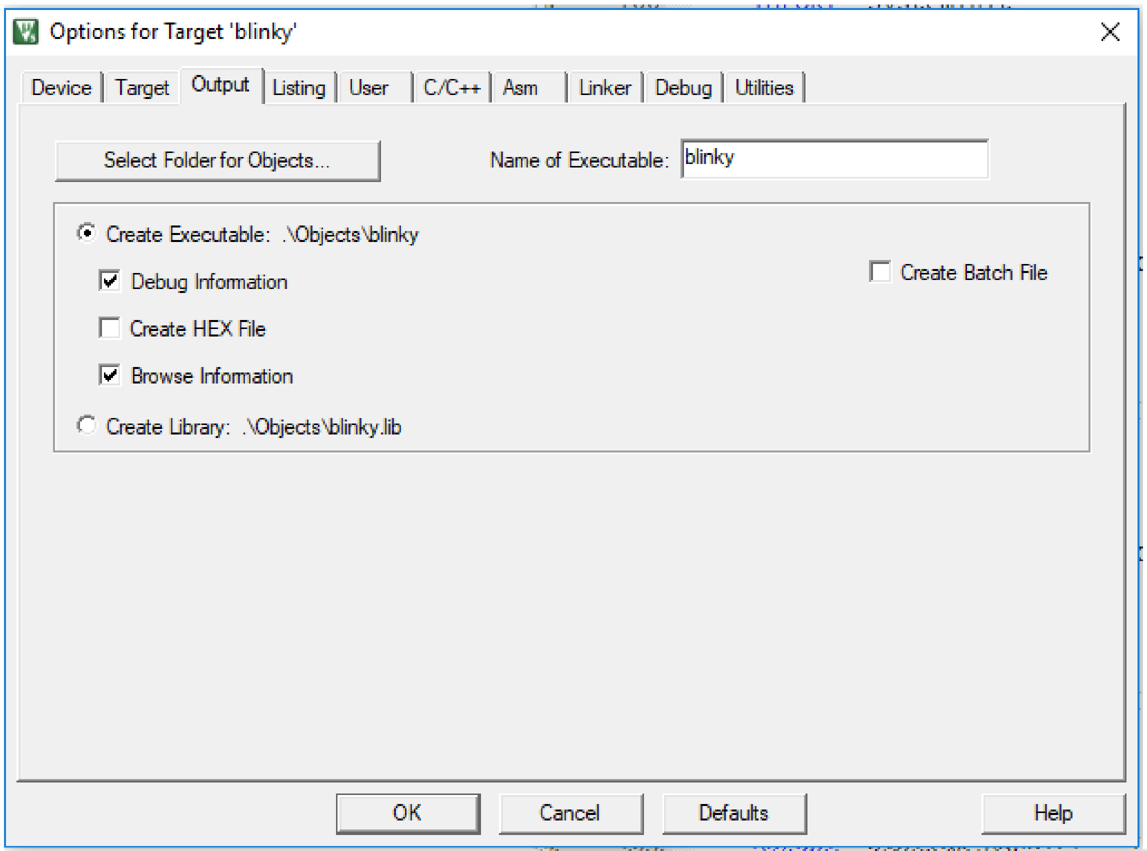
Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☐	IROM1:			○
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x0	0x1000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help


Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Select Folder for Objects...

Name of Executable: blinky

(●) Create Executable: .\Objects\blinky

[✓] Debug Information

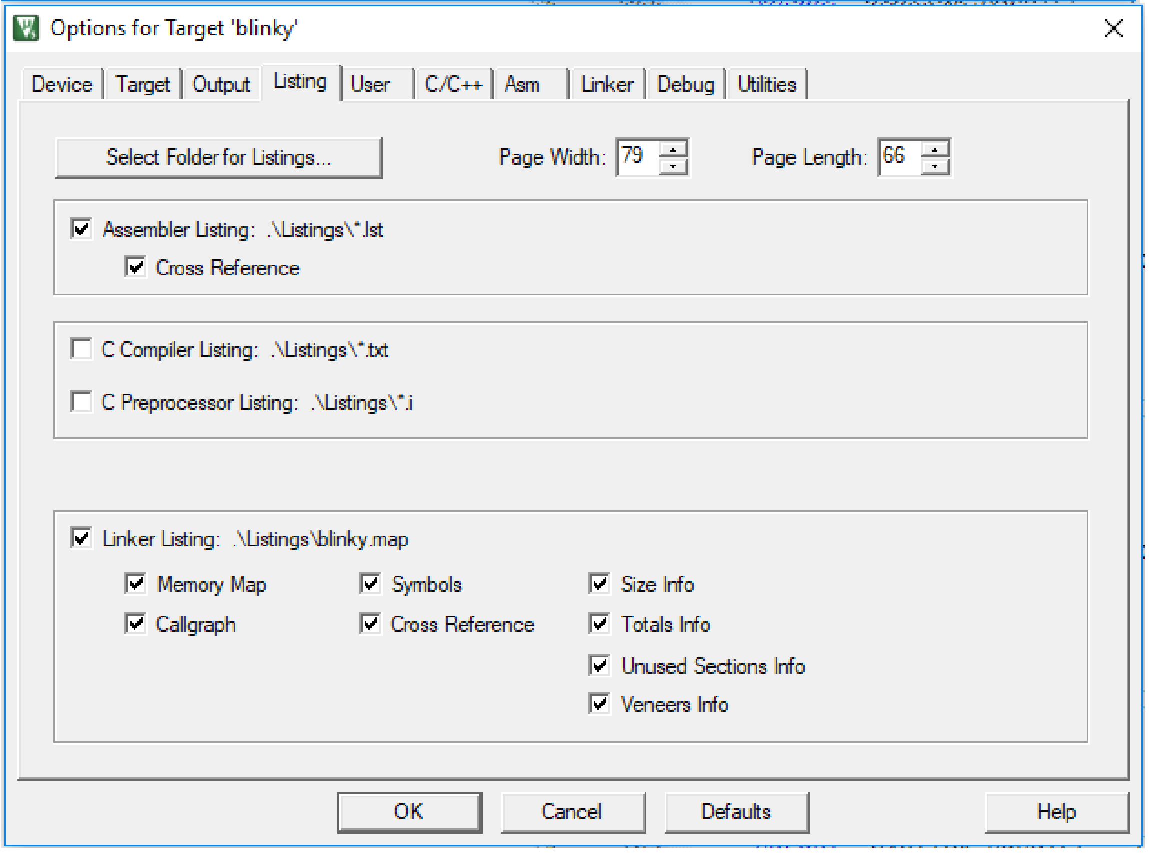
[ ] Create HEX File

[✓] Browse Information

( ) Create Library: .\Objects\blinky.lib

[ ] Create Batch File

OK | Cancel | Defaults | Help


Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Select Folder for Listings...

Page Width: 79

Page Length: 66

[✓] Assembler Listing: .\Listings*.lst

[✓] Cross Reference

[ ] C Compiler Listing: .\Listings*.txt

[ ] C Preprocessor Listing: .\Listings*.i

[✓] Linker Listing: .\Listings\blinky.map

[✓] Memory Map

[✓] Callgraph

[✓] Symbols

[✓] Cross Reference

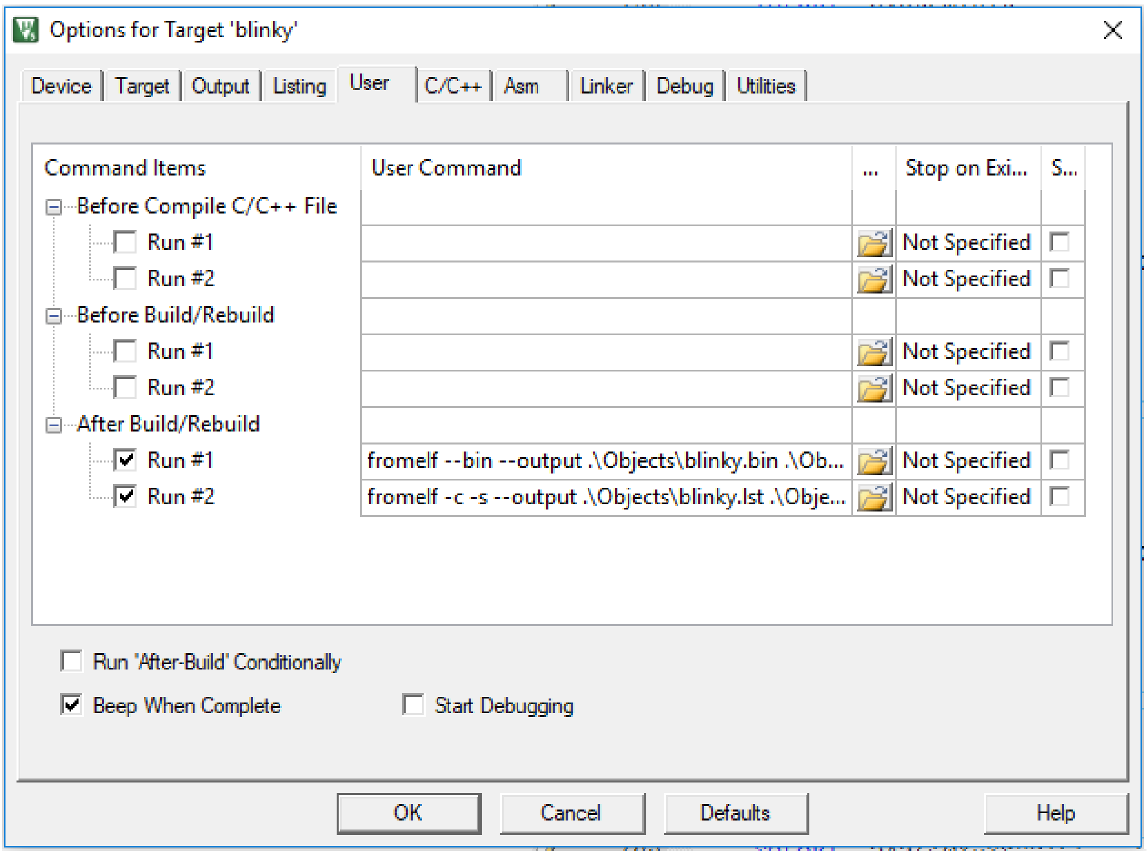
[✓] Size Info

[✓] Totals Info

[✓] Unused Sections Info

[✓] Veneers Info

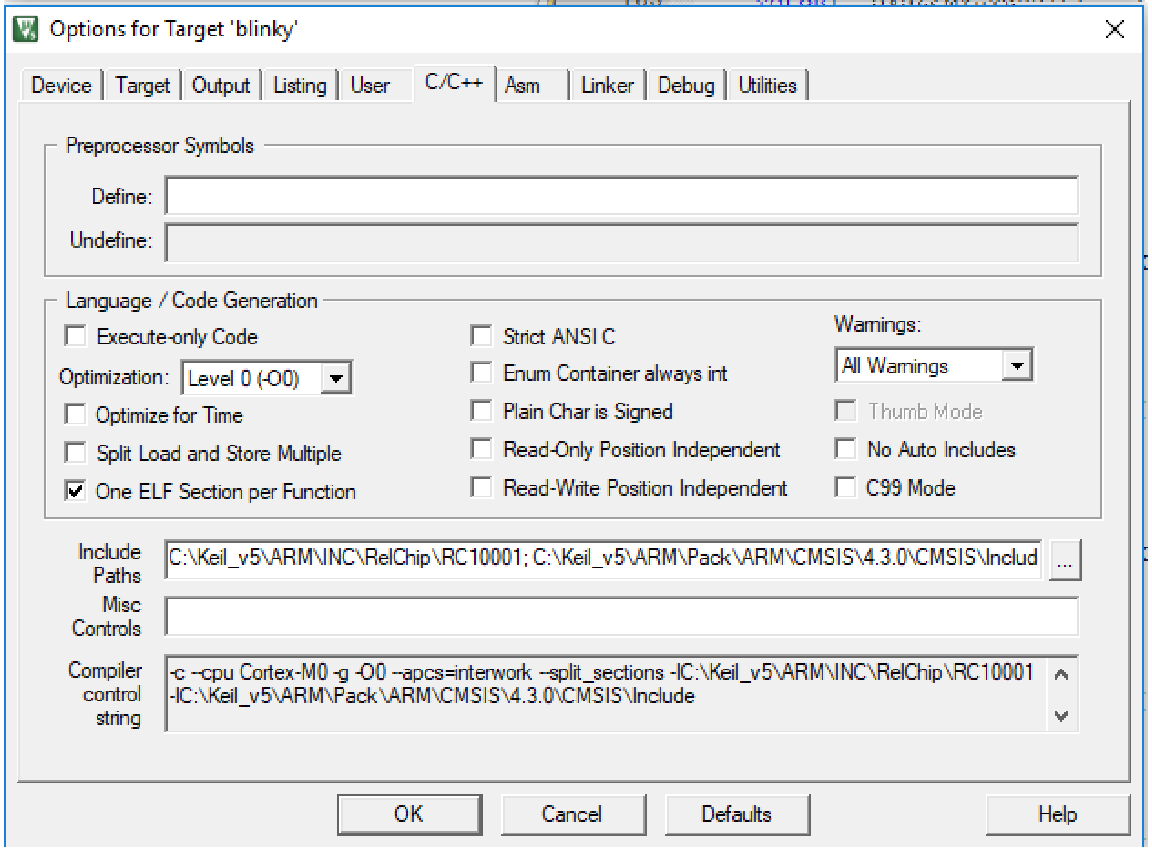
OK | Cancel | Defaults | Help


Options for Target 'blinky'

Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	fromelf --bin --output .\Objects\blinky.bin .\Ob...		Not Specified	☐
☑ Run #2	fromelf -c -s --output .\Objects\blinky.lst .\Obje...		Not Specified	☐

☐ Run 'After-Build' Conditionally

☑ Beep When Complete ☐ Start Debugging


Options for Target 'blinky'

Preprocessor Symbols

Define:

Undefine:

Language / Code Generation

☐ Execute-only Code

Optimization: Level 0 (-O0)

☐ Optimize for Time

☐ Split Load and Store Multiple

☑ One ELF Section per Function

☐ Strict ANSI C

☐ Enum Container always int

☐ Plain Char is Signed

☐ Read-Only Position Independent

☐ Read-Write Position Independent

Warnings: All Warnings

☐ Thumb Mode

☐ No Auto Includes

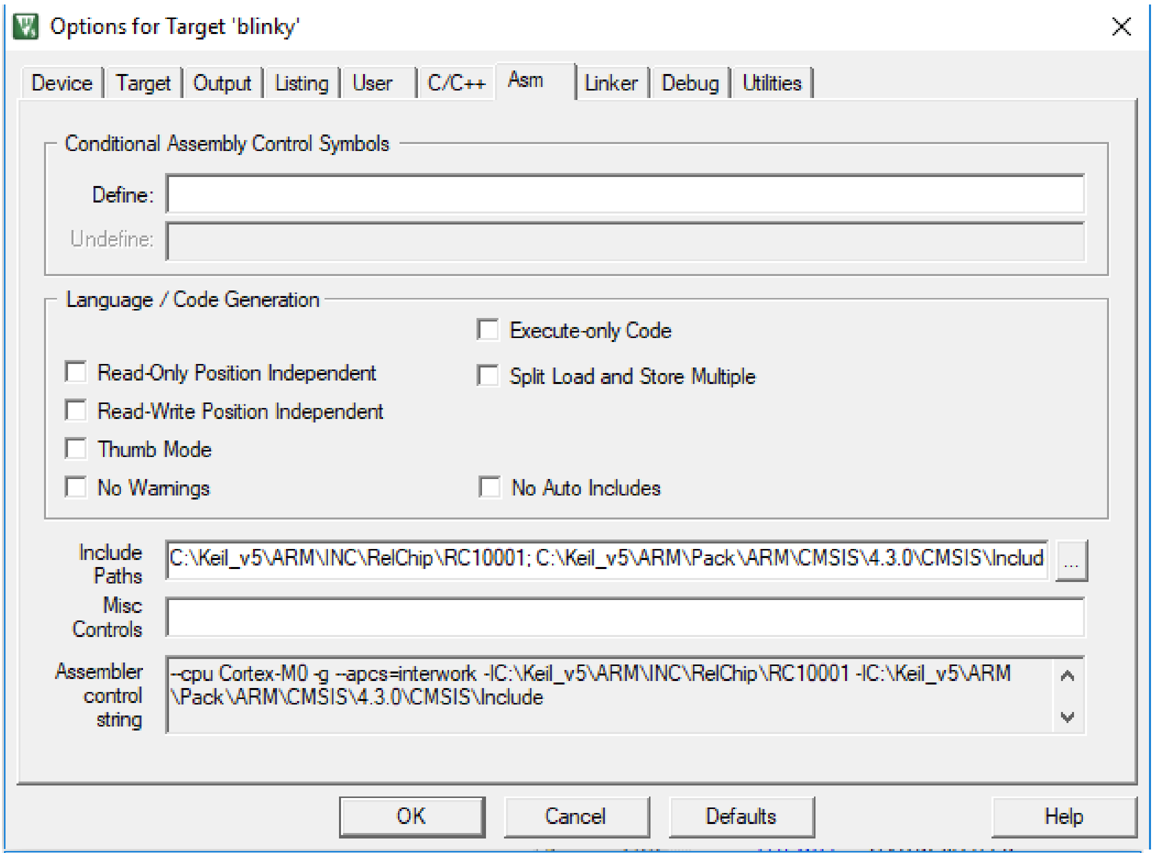
☐ C99 Mode

Include Paths: C:\Keil_v5\ARM\INC\RelChip\RC10001; C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.3.0\CMSIS\Includ

Misc Controls:

Compiler control string:

```
-c --cpu Cortex-M0 -g -O0 --apcs=interwork --split_sections -IC:\Keil_v5\ARM\INC\RelChip\RC10001
-IC:\Keil_v5\ARM\Pack\ARM\CMSIS\4.3.0\CMSIS\Include
```

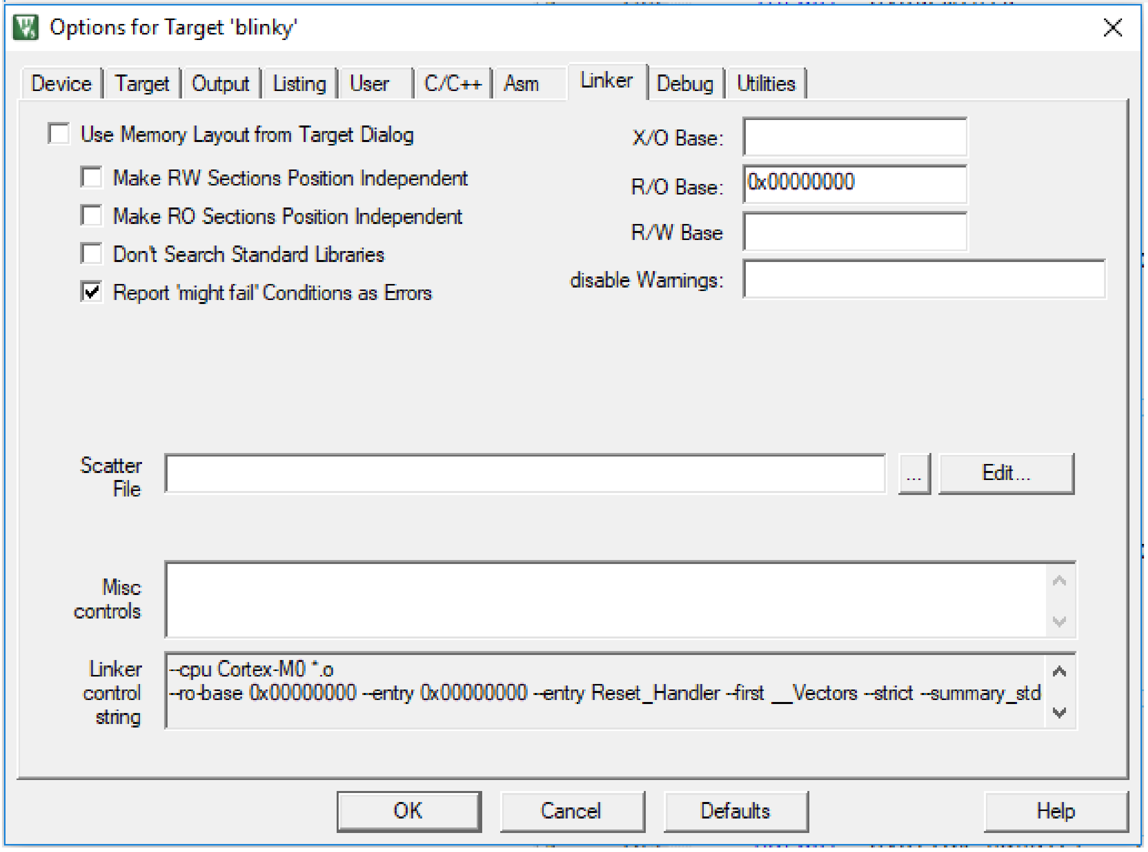


Options for Target 'blinky'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Conditional Assembly Control Symbols
Define:
Undefine:
Language / Code Generation
Execute-only Code
Read-Only Position Independent
Split Load and Store Multiple
Read-Write Position Independent
Thumb Mode
No Warnings
No Auto Includes
Include Paths
Misc Controls
Assembler control string
OK
Cancel
Defaults
Help

Include Paths: C:\Keil_v5\ARM\INC\RelChip\RC10001; C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.3.0\CMSIS\Includ

Assembler control string:

```
--cpu Cortex-M0 -g --apcs=interwork -IC:\Keil_v5\ARM\INC\RelChip\RC10001 -IC:\Keil_v5\ARM
\Pack\ARM\CMSIS\4.3.0\CMSIS\Include
```



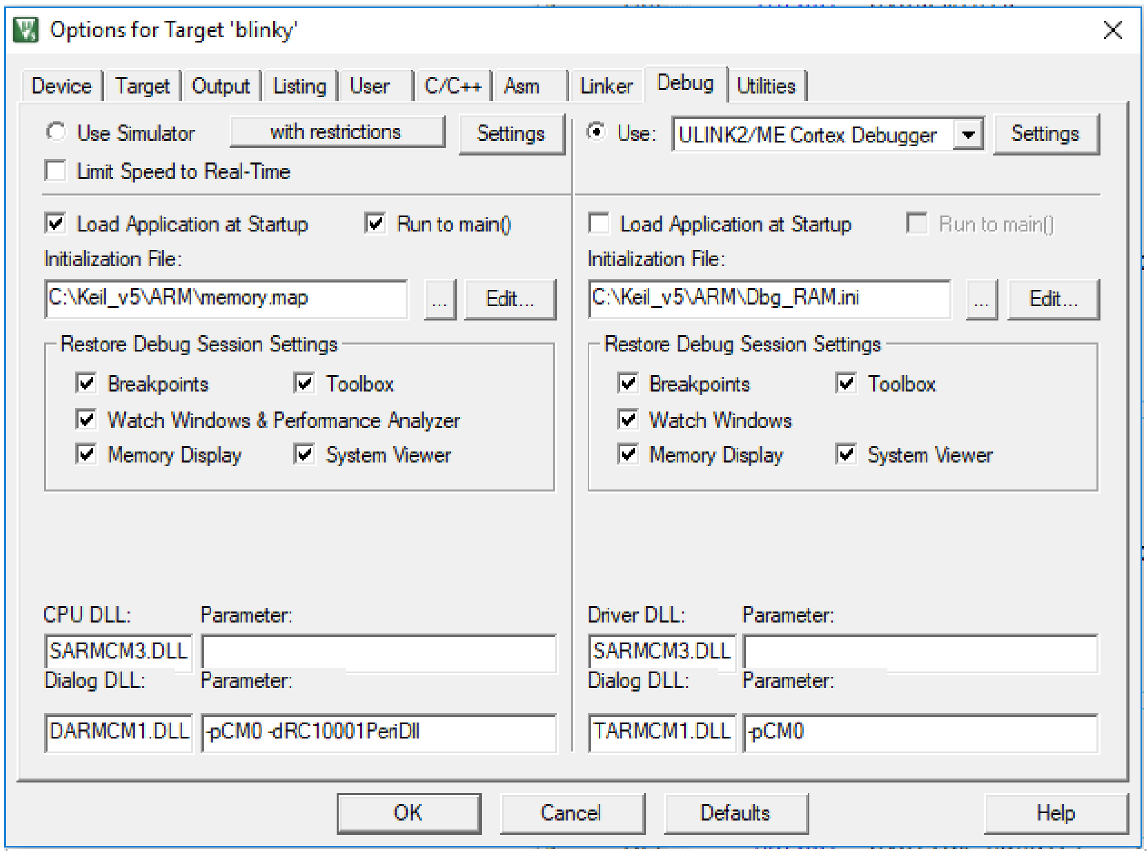
Options for Target 'blinky'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
X/O Base:
R/O Base:
R/W Base
disable Warnings:
Scatter File
Edit...
Misc controls
Linker control string
OK
Cancel
Defaults
Help

☐ Use Memory Layout from Target Dialog
☐ Make RW Sections Position Independent
☐ Make RO Sections Position Independent
☐ Don't Search Standard Libraries
☑ Report 'might fail' Conditions as Errors

R/O Base: 0x00000000

Linker control string:

```
--cpu Cortex-M0 *.o
--ro-base 0x00000000 --entry 0x00000000 --entry Reset_Handler --first __Vectors --strict --summary_std
```

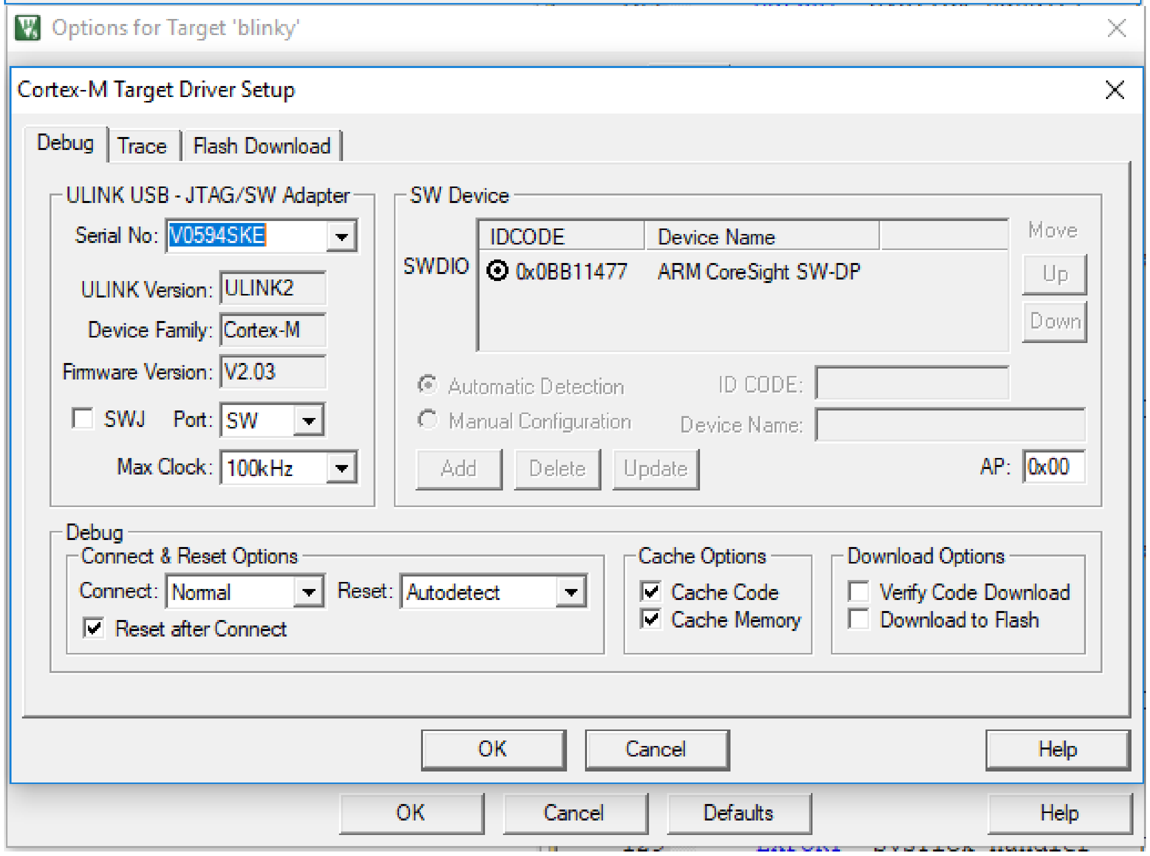

Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

○ Use Simulator with restrictions Settings
[ ] Limit Speed to Real-Time
[✓] Load Application at Startup [✓] Run to main()
Initialization File:
C:\Keil_v5\ARM\memory.map ... Edit...
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows & Performance Analyzer
[✓] Memory Display [✓] System Viewer
CPU DLL: SARMCM3.DLL Parameter:
Dialog DLL: DARMCM1.DLL Parameter: -pCM0 -dRC10001PeriDll

● Use: ULINK2/ME Cortex Debugger Settings
[ ] Load Application at Startup [ ] Run to main()
Initialization File:
C:\Keil_v5\ARM\Dbg_RAM.ini ... Edit...
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows
[✓] Memory Display [✓] System Viewer
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TARMCM1.DLL Parameter: -pCM0

OK Cancel Defaults Help


Cortex-M Target Driver Setup

Debug | Trace | Flash Download

ULINK USB - JTAG/SW Adapter
Serial No: V0594SKE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V2.03
[ ] SWJ Port: SW
Max Clock: 100kHz

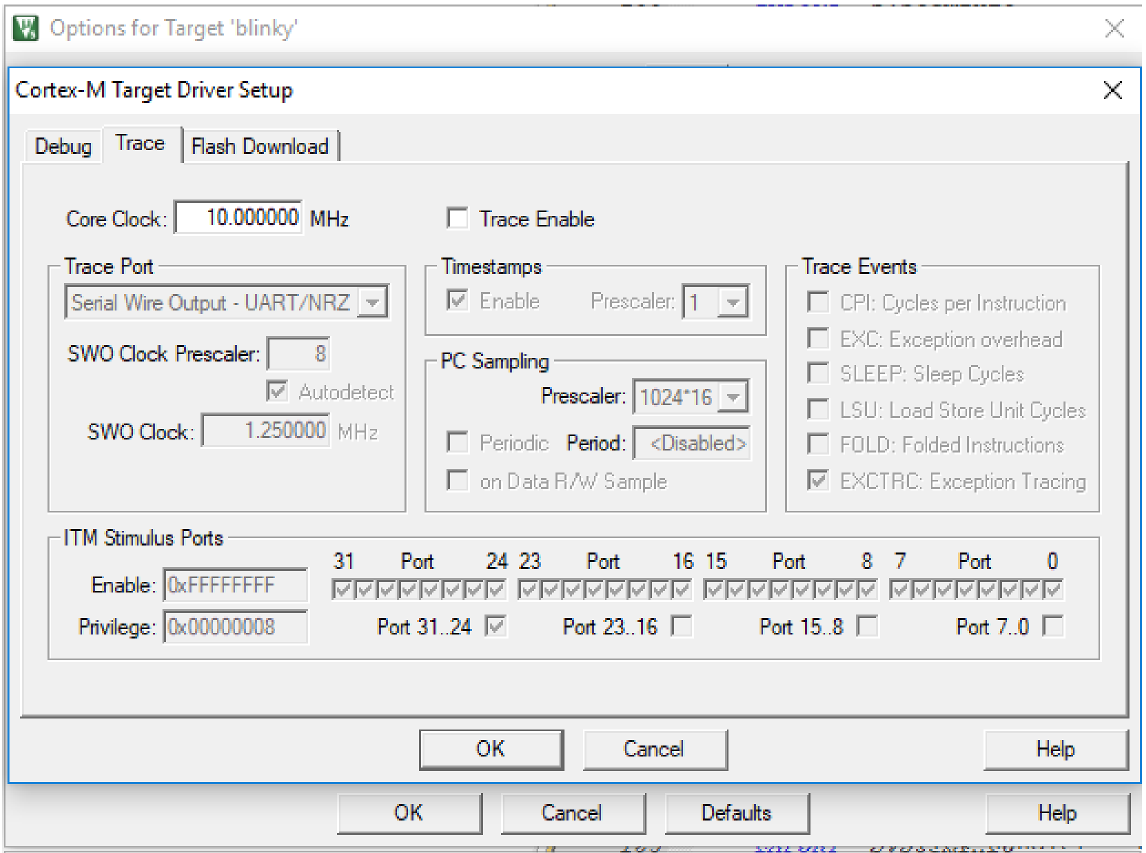
SW Device
	IDCODE	Device Name
SWDIO	● 0x0BB11477	ARM CoreSight SW-DP

Move Up Down
● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
Add Delete Update AP: 0x00

Debug
Connect & Reset Options
Connect: Normal Reset: Autodetect
[✓] Reset after Connect
Cache Options
[✓] Cache Code
[✓] Cache Memory
Download Options
[ ] Verify Code Download
[ ] Download to Flash

OK Cancel Help

OK Cancel Defaults Help



Options for Target 'blinky'

Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 10.000000 MHz

[ ] Trace Enable

Trace Port: Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 8

[✓] Autodetect

SWO Clock: 1.250000 MHz

Timestamps: [✓] Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16

[ ] Periodic Period: <Disabled>

[ ] on Data R/W Sample

Trace Events:

[ ] CPI: Cycles per Instruction

[ ] EXC: Exception overhead

[ ] SLEEP: Sleep Cycles

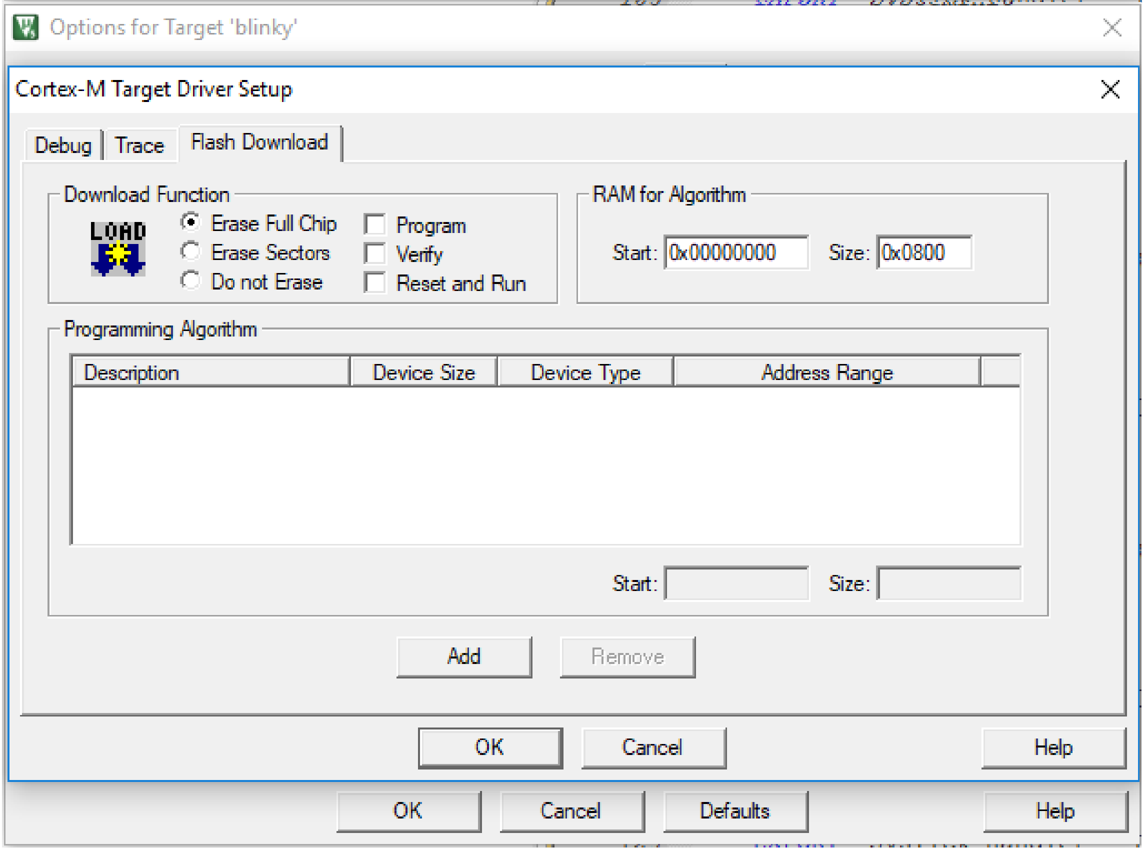
[ ] LSU: Load Store Unit Cycles

[ ] FOLD: Folded Instructions

[✓] EXCTRC: Exception Tracing

ITM Stimulus Ports

	31 Port 24	23 Port 16	15 Port 8	7 Port 0
Enable: 0xFFFFFFFF	✓✓✓✓✓✓✓✓	✓✓✓✓✓✓✓✓	✓✓✓✓✓✓✓✓	✓✓✓✓✓✓✓✓
Privilege: 0x00000008	Port 31..24 [✓]	Port 23..16 [ ]	Port 15..8 [ ]	Port 7..0 [ ]



Options for Target 'blinky'

Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Download Function

(●) Erase Full Chip [ ] Program

( ) Erase Sectors [ ] Verify

( ) Do not Erase [ ] Reset and Run

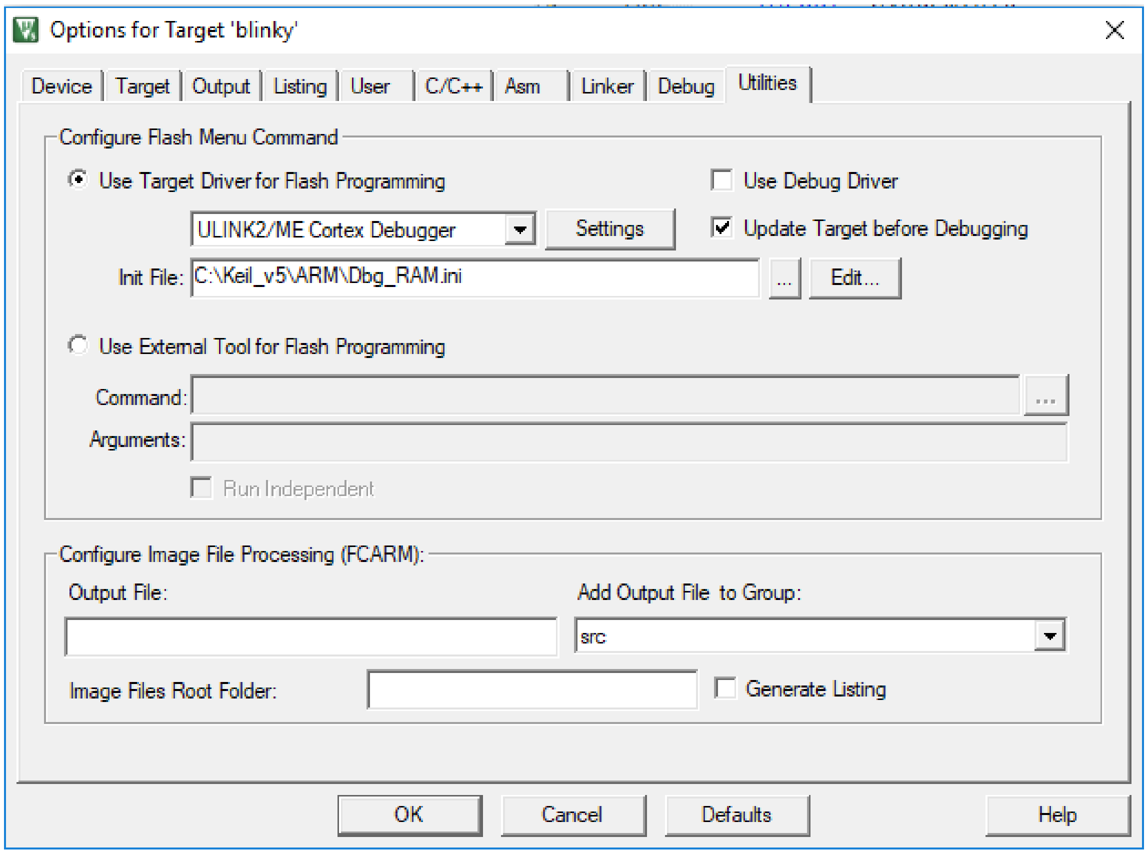
RAM for Algorithm

Start: 0x00000000 Size: 0x0800

Programming Algorithm

Description	Device Size	Device Type	Address Range

Start: Size:



Options for Target 'blinky'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Configure Flash Menu Command

(●) Use Target Driver for Flash Programming

☐ Use Debug Driver

ULINK2/ME Cortex Debugger | Settings

☑ Update Target before Debugging

Init File: C:\Keil_v5\ARM\Dbg_RAM.ini | ... | Edit...

( ) Use External Tool for Flash Programming

Command:

Arguments:

☐ Run Independent

Configure Image File Processing (FCARM):

Output File:

Add Output File to Group: src

Image Files Root Folder:

☐ Generate Listing

OK | Cancel | Defaults | Help

A.3. RC10001 GPIO Pulse Code

RC10001 example blink code was available in the Keil library set (MDK v4 Legacy Support).

Main C code: "RC10001 Blink.c"

```
/*****************************************************************************
 ****************************************************************************
 **
 **     File:       blinky.c
 **     Author:  RelChip, Inc.
 **     Date:      August 10, 2015
 **
 ** Copyright(c) 2015 RelChip, Inc. All Rights Reserved except
 ** as granted below.
 **
 **     Description:
 **     A demonstration program to display a sequence of LED
 ** lights on the development board. The program is stored
 ** stored internally and boot loaded with the Keil serial debug
 ** port.
 **     The LEDs sequence in a counter fashion. Pressing the
 ** GPIO3_0 switch will slow down the LED sequence, while
 ** pressing the GPIO2_11 switch will increase the LED
 ** sequence speed.
 **
 ** THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES,
 ** WHETHER EXPRESS, IMPLIED OR STATUTORY,
 ** INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES
 ** OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 ** PURPOSE APPLY TO THIS SOFTWARE. RELCHIP SHALL
 ** NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
 ** INCIDENTAL,  OR CONSEQUENTIAL DAMAGES, FOR ANY
 ** REASON WHATSOEVER.
 **
 ** LICENSE IS GRANTED TO USE THIS CODE FREELY AT
 ** YOUR OWN RISK.
 **
 ******************************************************************************
 ******************************************************************************
 */

#include "RC10001.h"                    /* RC10001 definitions */

uint32_t LoopCount;                     /* Delay Count */

/******************************************************************************
 ******************************************************************************
 **
 **     Interrupt Routines
 **
 */
/******************************************************************************
 ******************************************************************************
 **
 **     GPIO3 Interrupt
 **
 ** Description:
 **     Speeds LED Frequency Up
 ** Input: None
 ** Return: None
 **
 */
void GPIO2_IRQHandler(void)
{
  RC_GPIO2->ICR = 0xfff;                /* Clear Interrupts */
  LoopCount = LoopCount >> 1;      /* Count Divide by 2 */
}
/******************************************************************************
 ******************************************************************************
 **
 **     GPIO3 Interrupt
 **
 ** Description:
 **     Slows LED Frequency Down
 ** Input: None
 ** Return: None
 **
 */
void GPIO3_IRQHandler(void)
{
  RC_GPIO3->ICR = 0xfff;                /* Clear Interrupts */
  LoopCount = LoopCount << 1;      /* Count Times 2 */
}

/*****************************************************************************
 *****************************************************************************
 **
 **     Main
 **
 */
int main (void) {

  uint32_t i;
          uint16_t count = 0;

  LoopCount = 0;
  RC_GPIO2->ICR = 0xfff;                /* Clear Interrupts */
  RC_GPIO3->ICR = 0xfff;                /* Clear Interrupts */
                                       /* */
  /* Infinite Loop */
  while(1) {
    /* Wait with Dummy */
                    count++;
                    if( count >= 1000 )
                    {
                              count = 0;
                              /* Watch out that this does not get optimized out */
                              for ( i = 0; i < LoopCount; i++ )  { i=i; }
                              RC_GPIO0->MASKED_ACCESS [0xff] =
                                        RC_GPIO0->MASKED_ACCESS[0xff] + 1;
                    }
  }
}
```

Peripheral C code: "system_RC10001.c"

```
/*****************************************************************************
 *****************************************************************************
 *
 * @file          system_RC10001.h
 *
 * @brief         CMSIS Cortex-M0 Device Peripheral Access Layer Source File for
 *                the RelChip RC10001 Device
 *
 * @version       V0.20
 * @date          02 November 2014
 *
 * @note          Copyright (C) 2011-2014 RelChip, Inc. All Rights Reserved.
 *
 * @par
 * THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS,
IMPLIED,
 * OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 * RELCHIP SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 * OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 ******************************************************************************
 ******************************************************************************/
#define __SYSTEM_RC10001_C

#include <stdint.h>
#include "RC10001.h"

#define CLOCK_SETUP             1
#define SYSAHBCLKDIV_Val        1       // Reset: 0x001

/*----------------------------------------------------------------------------
  Check the Register Settings
 *----------------------------------------------------------------------------*/
#define CHECK_RANGE(val, min, max)      ((val < min) || (val > max))
#define CHECK_RSVD(val, mask)    (val & mask)

#if (CHECK_RANGE((SYSAHBCLKDIV_Val), 0, 255))
   #error "SYSAHBCLKDIV: Value out of range!"
#endif

/*----------------------------------------------------------------------------
  DEFINES
 *----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
  Define Clocks
 *----------------------------------------------------------------------------*/
#define __XTAL                  (3686400UL)    /* Oscillator Frequency */
```

```
#define __SYS_OSC_CLK      (    __XTAL)     /* Main Oscillator Frequency */
#define __SYSTEM_CLOCK (__SYS_OSC_CLK)

/*----------------------------------------------------------------------------
  Clock Variable Definitions
 *----------------------------------------------------------------------------*/
uint32_t SystemCoreClock = __SYSTEM_CLOCK; /*!< System Clock Frequency */

/**
 *
 * Initialize the System
 *
 * @param       none
 * @return      none
 *
 * @brief       Initialize the Microcontroller System. Substitute for this
 *          code to configure the system before entering "main"
 *              Initialize the System.
 */
void SystemInit (void) {
    RC_GPIO0->DIR = 0xFF;          /* LED Output */
  RC_GPIO0->DATA = 0x00;         /* LED ON */

  /* GPIO2 Interrupt */
  RC_GPIO2->IS = 0;              /* Set Low Going Edge */
  RC_GPIO2->IEV = 0;
  RC_GPIO2->IE = 0x800;          /* Enable 2_11 */

  /* GPIO3 Interrupt */
  RC_GPIO3->IS = 0;              /* Set Low Going Edge */
  RC_GPIO3->IEV = 0;
  RC_GPIO3->IE = 0x1;            /* Enable 3_0 */

  /* NVIC Enable */
  NVIC_EnableIRQ(GPIO2_IRQn);
  NVIC_EnableIRQ(GPIO3_IRQn);
}
```

A.4. RC2110836 Register Alternating Code

Code used to evaluate the RC2110836 was written in MATLAB and interfaced via the NUC100VD3AN microcontroller.

RAM_Data_Analysis.m

```
% This script is for reading comma delimited files from the RC2110836 HT
% RAM device
```

```
clear all;
close all;

%Read in the Original data file which the new data will be compared to
Filename1 = 'C:\Users\atcashi\Documents\Component Testing\Relchip Ram
Tests\Ram_Oven_Data\RAM_Chip 1\250C_8-29-14_458PM_9-2-14_857AM_29mA.txt';

Original_data = dlmread(Filename1);

% % %Cut off the initial Errors
 Crop_Data = Original_data;
% % while (Crop_Data(1,1) ~= 170 || Crop_Data(1,2) ~= 170 || Crop_Data(1,3) ~= 170 ||
Crop_Data(1,4) ~= 170) && (Crop_Data(1,1) ~= 85 || Crop_Data(1,2) ~= 85 || Crop_Data(1,3)
~= 85 || Crop_Data(1,4) ~= 85)
% %     Crop_Data(1,:) = [];
% % end

%Extract Error Array
error_rows = find(Crop_Data(:,1) == 0 & Crop_Data(:,2) == 0 & Crop_Data(:,3) == 0 &
Crop_Data(:,4) == 0);

ErrorCount = length(error_rows)/2;

ind = 1;
for i = 1:length(error_rows)

   if sum(Crop_Data(error_rows(i)+1 ,1:4)) == 1020 && sum(Crop_Data(error_rows(i)+4 ,1:4))
== 0 && sum(Crop_Data(error_rows(i)+5 ,1:4)) == 1020
       Errors(ind,1) = error_rows(i);  %Errors will be formatted as
[Row,Address1,Address2,Address3,Address4,received1,received2,received3,received4,timestamp]
       Errors(ind,2) = Crop_Data(error_rows(i) + 2, 1); %Address1
       Errors(ind,3) = Crop_Data(error_rows(i) + 2, 2); %Address2
       Errors(ind,4) = Crop_Data(error_rows(i) + 2, 3); %Address3
       Errors(ind,5) = Crop_Data(error_rows(i) + 2, 4); %Address4
       Errors(ind,6) = Crop_Data(error_rows(i) + 3, 1); %Received1
       Errors(ind,7) = Crop_Data(error_rows(i) + 3, 2); %Received1
       Errors(ind,8) = Crop_Data(error_rows(i) + 3, 3); %Received1
       Errors(ind,9) = Crop_Data(error_rows(i) + 3, 4); %Received1
       Errors(ind,10) = Crop_Data(error_rows(i), 5); %Timestamp
       ind = ind + 1;
   end

end

Error_Timestamps_Str = datestr(Errors(:,10));

%Calculate bit error rate
for i = 1:length(Errors)
```

```
  fir = dec2bin(Errors(i,6));

  for g = 1:length(fir)
  end
  sec = dec2bin(Errors(i,6));
  thi = dec2bin(Errors(i,6));
  fou = dec2bin(Errors(i,6));

end
```

Log_SerialData.m

```
clear all;
close all;

if instrfind ~= 0
fclose(instrfind);
delete(instrfind);
end

LogFileName = 'TestLogging.txt';
%fopen('DataLog.txt');

Baud = 4420;
Bits = 8;

SerPort = serial('COM5', 'BaudRate', Baud, 'DataBits', Bits);

fopen(SerPort);

byteA = 0;
while byteA ~= 47
 byteA = fread(SerPort, 1);
end

byteB = 0;
while byteB ~= 110
 byteB = fread(SerPort, 1);
end

true = 1;
row = 1;
while true == 1
%Read in a line of data
 byte1 = fread(SerPort, 1);
 byte2 = fread(SerPort,1);
 byte3 = fread(SerPort, 1);
 byte4 = fread(SerPort,1);
 byte5 = fread(SerPort, 1);
 byte6 = fread(SerPort,1);

 if byte5 ~= 47 || byte6 ~= 110
    disp('Data is staggered');
    break;
 end

  DataLine = [byte1, byte2, byte3, byte4];
  dlmwrite(LogFileName, DataLine,'-append');

end
```

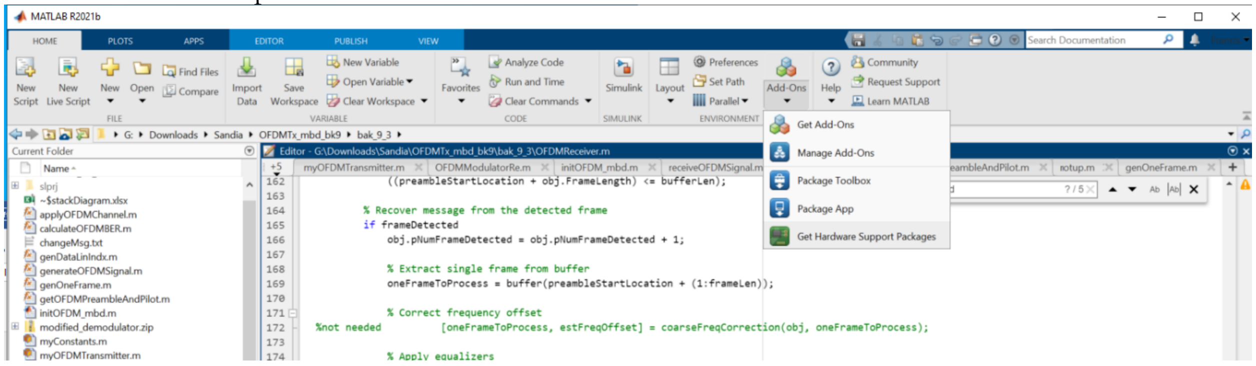
A.5. OFDM Tx_mbd: Procedure to modify the C code and sample the data for the SM320F28335-HT data link

Release version:	10
Date:	25th Feb./ 2022
Author:	Francis Tiong, ftiong@mathworks.com

Contents

Introduction
This document describes how to run the code in OFDMTx_mbd and how it can be used to generate embedded code for a target processor. The project is intended to send OFDM signals through a hardware board with TI Delfino F2833x. The chip was selected since it can operate in high temperature. Further, the modified C section would describe how to modify the C code using the Code Composer. The section sampling the data through Picoscope would describe the procedure to sample the data.
Prerequisite – Installing Texas Instruments C2000 support package
1, In MATLAB, on the top row select the “HOME” tab.

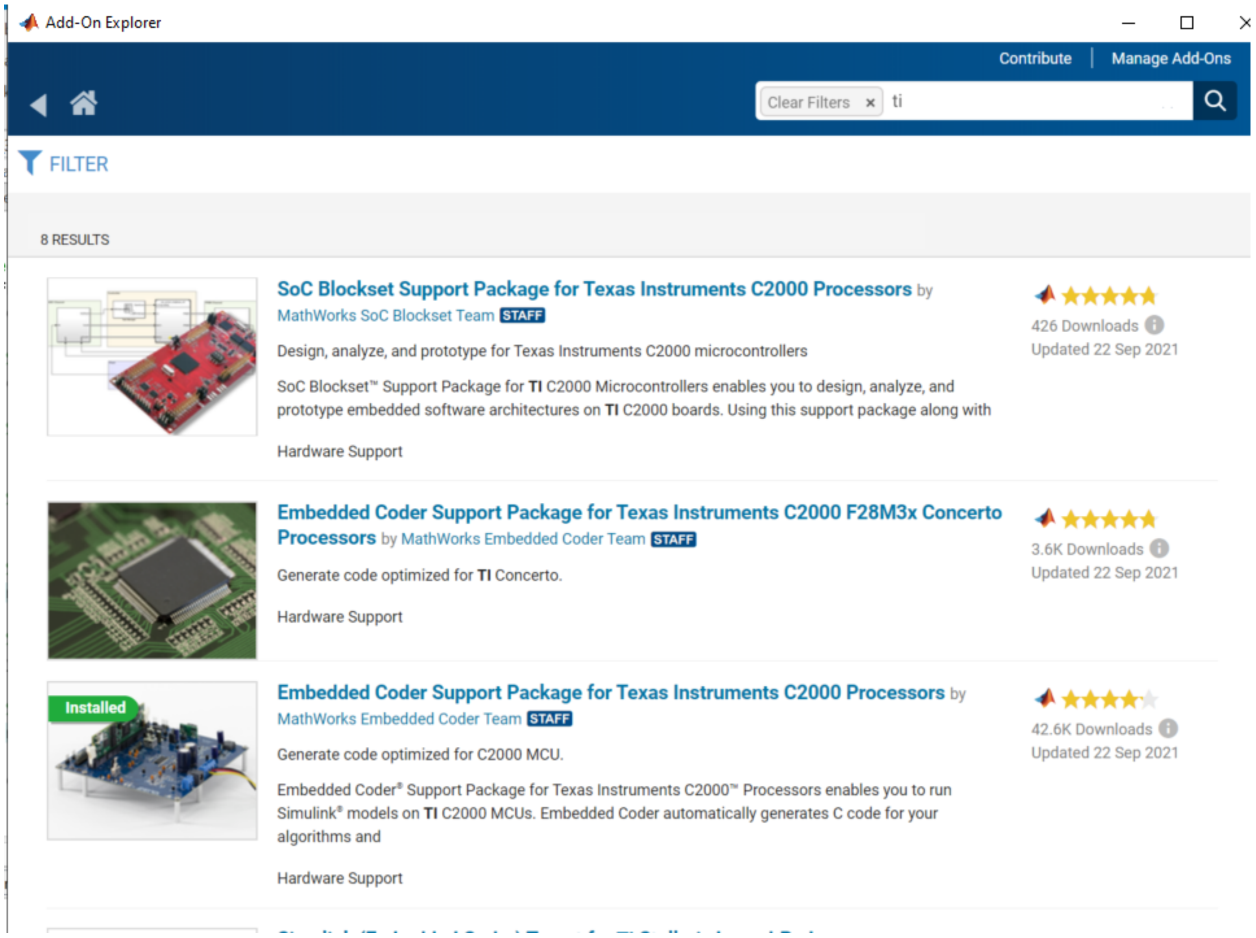
2, On the top bar look for the tri-color cubes with the word "Add-Ons" and click on the down arrow to see the drop down manual.



Editor - G:\Downloads\Sandia\OFDMTx_mbd_bk9\bak_9_3\OFDMReceiver.m

```
            ((preambleStartLocation + obj.FrameLength) <= bufferLen);

        % Recover message from the detected frame
        if frameDetected
            obj.pNumFrameDetected = obj.pNumFrameDetected + 1;

            % Extract single frame from buffer
            oneFrameToProcess = buffer(preambleStartLocation + (1:frameLen));

            % Correct frequency offset
%not needed          [oneFrameToProcess, estFreqOffset] = coarseFreqCorrection(obj, oneFrameToProcess);

            % Apply equalizers
```

3, At the drop down manual select "Get Hardware Support Packages"
4, In the Add-On Explorer search box type "TI".
5, Select "Embedded Coder Support Package for Texas Instruments C2000 Processors".



8 RESULTS

SoC Blockset Support Package for Texas Instruments C2000 Processors by MathWorks SoC Blockset Team STAFF

Design, analyze, and prototype for Texas Instruments C2000 microcontrollers

SoC Blockset™ Support Package for **TI** C2000 Microcontrollers enables you to design, analyze, and prototype embedded software architectures on **TI** C2000 boards. Using this support package along with

Hardware Support

426 Downloads
Updated 22 Sep 2021

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors by MathWorks Embedded Coder Team STAFF

Generate code optimized for **TI** Concerto.

Hardware Support

3.6K Downloads
Updated 22 Sep 2021

Installed

Embedded Coder Support Package for Texas Instruments C2000 Processors by MathWorks Embedded Coder Team STAFF

Generate code optimized for C2000 MCU.

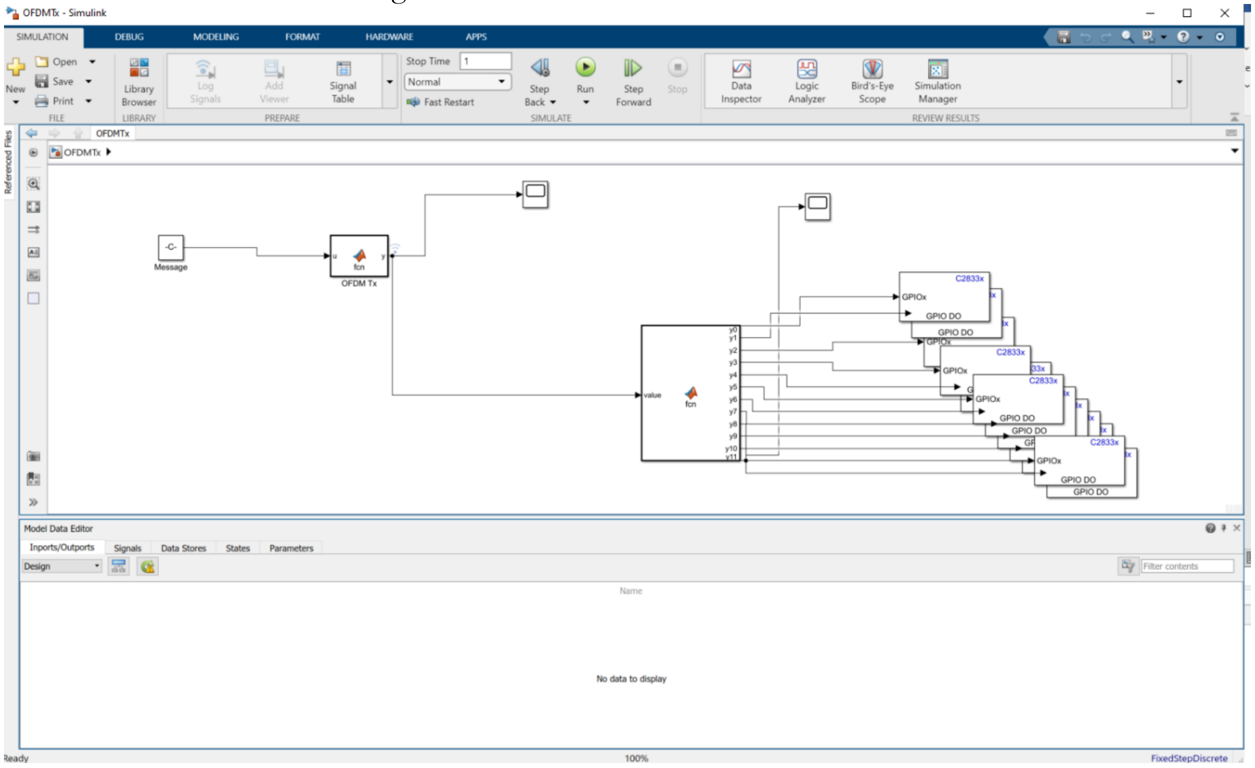
Embedded Coder® Support Package for Texas Instruments C2000™ Processors enables you to run Simulink® models on **TI** C2000 MCUs. Embedded Coder automatically generates C code for your algorithms and

Hardware Support

42.6K Downloads
Updated 22 Sep 2021

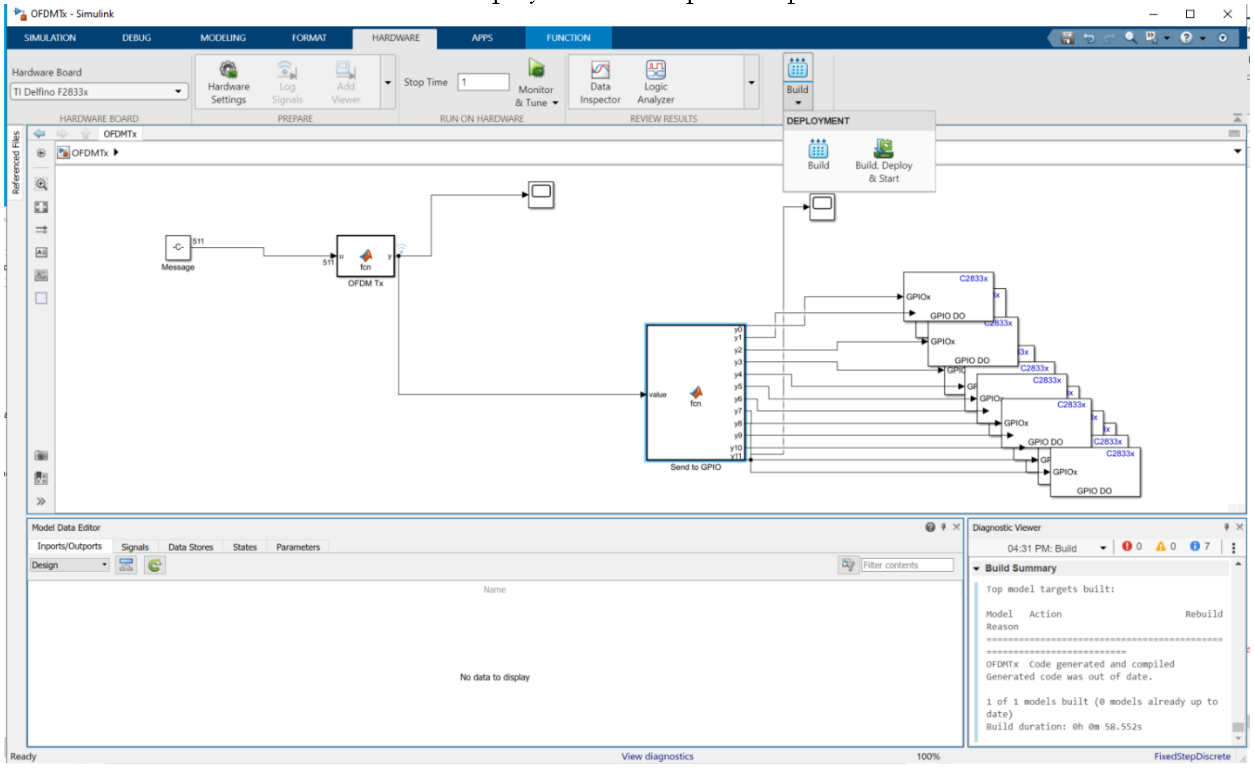
Instructions to run the code in MATLAB
Note that in MATLAB one can simulate the transmitter, the propagation channel and the receiver. This is a good way to check if the transmitter code is working correctly.
1, unzip the code "OFDMTx_mbd_9_3_02082022.zip" into a folder. From now on we will assume the unzipped files are in a folder called "OFDMTx_mbd_9_3_02082022".

2, In MATLAB, change the directory of the current workspace to be in the folder "OFDMTx_mbd_9_3_02082022".
>>cd G:\Downloads\Sandia\OFDMTx_mbd_9_3_02082022
3, In MATLAB, run the script "OFDMSynchronizationExample.m"
>> OFDMSynchronizationExample
4, If one would like to run the whole system again including the constructors then it is necessary to clear the persistent variables defined first before running.
>>clear classes
>> OFDMSynchronizationExample
Instructions to simulate the Tx code in Simulink
Note that in Simulink one can simulate the transmitter. The transmitted signal can be observed in the scope that is connected to the OFDM Tx block.
1, In MATLAB, change the directory of the current workspace to be in the folder "OFDMTx_mbd_9_3_02082022".
>>cd G:\Downloads\Sandia\OFDMTx_mbd_9_3_02082022
2, click on the file "OFDMTx.slx" from the folder window or enter "OFDMTx"
>>OFDMTx
3, On the top left corner of the Simulink window select the tab "SIMULATION". At the top row around the middle click on the green "Run" button.



Instructions to generate/run code for the target board in Simulink
Like simulating the signal one can select the build function under the hardware tab to generate the target embedded code. A code generation report will pop out upon completion.
1, In MATLAB, change the directory of the current workspace to be in the folder "OFDMTx_mbd_9_3_02082022".
>>cd G:\Downloads\Sandia\OFDMTx_mbd_9_3_02082022

2, click on the file “OFDMTx.slx” from the folder window or enter “OFDMTx”
\>>OFDMTx
3, On the top row of the Simulink window select the tab “HARDWARE”. At the top right end click on the drop down arrow to see the Manuel under “Build”.
4, click on the “Build” button to generate the target code. If it is desired to build, download and run the code on the board then one can click on “Build Deploy & Start” instead.
5, While the coding is being compiled one can click on the words “View diagnostics” at the very bottom of the screen to see verbose display of the compilation process.



```
Top model targets built:

Model   Action                          Rebuild
Reason
==============================================
=========================
OFDMTx  Code generated and compiled
Generated code was out of date.

1 of 1 models built (0 models already up to
date)
Build duration: 0h 0m 58.552s
```

Brief description of the files included
It seems this OFDM project was originated from an OFDM example from MathWorks -- https://www.mathworks.com/help/comm/ug/ofdm-synchronization.html. The files provided in the zip file “OFDMTx_mbd_9_3_02082022” have been heavily optimized. There are comments that resides with the code that will explain what the variables and operations are intending to achieve. A stack diagram illustrating the relationship between files is shown below:

OFDMSynchronizationExample.m	myConstants.m			
	initOFDM_mbd.m	getOFDMPreambleAndPilot.m		
		rotup.m		
	generateOFDMSignal.m	myOFDMTransmitter.m	genDataLinIndx.m	
			OFDMModulatorRe.m	ofdmModulateRe_single.m
	applyOFDMChannel.m			
	rotDown.m			
	receiveOFDMSignal.m	OFDMReceiver.m		
	calculateOFDMBER.m			

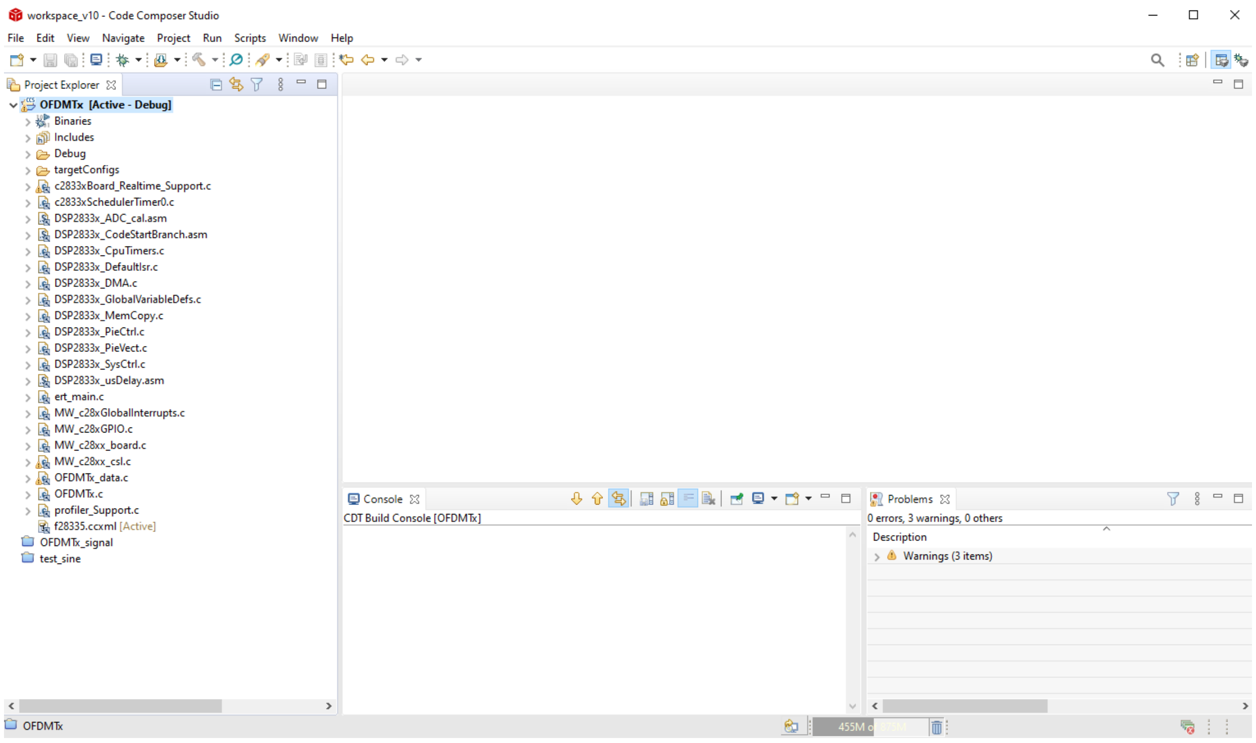
From the diagram, the caller files are on the left and the callee is on the right. For example, the file initOFDM_mbd.m will call two files – getOFDMPreambleAndPilot and rotup. The file initOFDM_mbd is called by OFDMSucrhoizationExample.

OFDMSynchronizationExample.m --- This is the main calling script for MATLAB. Inside this file one can also find documented description of the overall system. It will prepare a message to be sent, generate an OFDM signal, simulate the signal going through a propagation channel, demodulate the received signal, recovered the message sent and then calculate the error rate. Note that comparing to the original code the signal now being sent into applyOFDMChannel has now gone through a Hilbert transform first in order to obtain the complex equivalent values before applying the channel distortion.
myConstants.m – This is a constant class structure that contains important constants that are used throughout the simuation.
initOFDM_mbd.m – This script initiates a message to be sent out, creates an array for the preamble sequence and an array for the pilot symbols. These fixed data are to be generated during construction time to reduce code space and execution time. This file is also used in the Simulink model.
getOFDMPreambleAndPilot.m – this function generates the preamble sequence and the pilot symbols needed. In code generation the generated data would be used, and this code would not be ported into program code.
rotup.m – this code would double the sampling frequency and a complex rotation of the input. This code would not be generated as program code.
generateOFDMSignal.m – This is the entry point of the transmitter. This code is used in both he Matlab simulation as well as in the Simulink. All this function does is to create an instance of the class myOFDMTransmitter.
myOFDMTransmitter.m – This code has been highly optimized to reduce memory and cycles usage. The minimal buffering memory needed would be to have the code generates one IFFT data frame per call. Thus, every time this code is called it will generate one IFFT data frame (or 160 samples). In the stepImpl function there is a state machine that depends on the frameIdx value, a different data would be used to generate the OFDM signal. Each message would be transmitted with the preamble sequence first followed by a message content. The preamble would take 4 IFFT data frames to transmit and the message will take 11 IFFT data frames. The preamble is precalculated as stored in an array while the message would be converted as BPSK complex payload bits and then pass into OFDMModulatorRe.
genDataLinIndx.m – This is a function that generates an index table. The table would indicate which FFT bin would be used to transmit message data. This function currently resides in the constructor and thus it would not be code generated. The table would be stored and used instead. This code can be moved to be used during run-time if there is a need to reduce the memory usage.
OFDMModulatorRe.m – This function does the loading of the message bits as well as the pilot symbols onto the FFT bins. After loading the complex array would be send to ofdmModulateRe_single.
ofdmModulateRe_single.m – this function does the IFFT, up-rotation and the appending of cyclic prefix to generate the OFDM signal. Note that the up-rotation is done through doubling of the IFFT bins. A real (non-complex) signal at the output of the IFFT is ensured when the complex frequency domain signal is complex conjugate.
applyOFDMChannel.m – There is a slight modification to the original code. Here the frequency offset has been scaled according to an input value – freqOffPPM. The input freqOffPPM is the

parts-per-million offset which serves as the PPM offset between the transmitter and the receiver oscillators. Note that the input to this function is expected to be complex valued.
rotDown.m -- this code would do complex rotation down and then down sample by half. This code is being used only the receiver.
receiveOFDMSignal.m – this function creates an instance of the class OFDMReceiver.
OFDMReceiver.m – This is the main code for the receiver. It calls the function "locatePreamble", "frameEqualization", "comm.OFDMDemodulator" and then "comm.BPSKDemodulator". Some optimization has been done to simplify the code structure. In addition, the function coarseFreqCorrection has been disabled. It was found that the function is simply not needed in this application. Even with high clock rate difference the pilot tone tracking method to be applied in the frameEqualization function is enough to mitigate the effect. In the frameEqualization function the interpolation of the phase error based on the pilot tones onto the data bins are to be done through the interpolation function and not the resample function as in the original code. More descriptions of the receiver can be found inside OFDMSynchronizationExample.m.
calculateOFDMBER.m – This function calculates the bit error rate and the frame error rate.

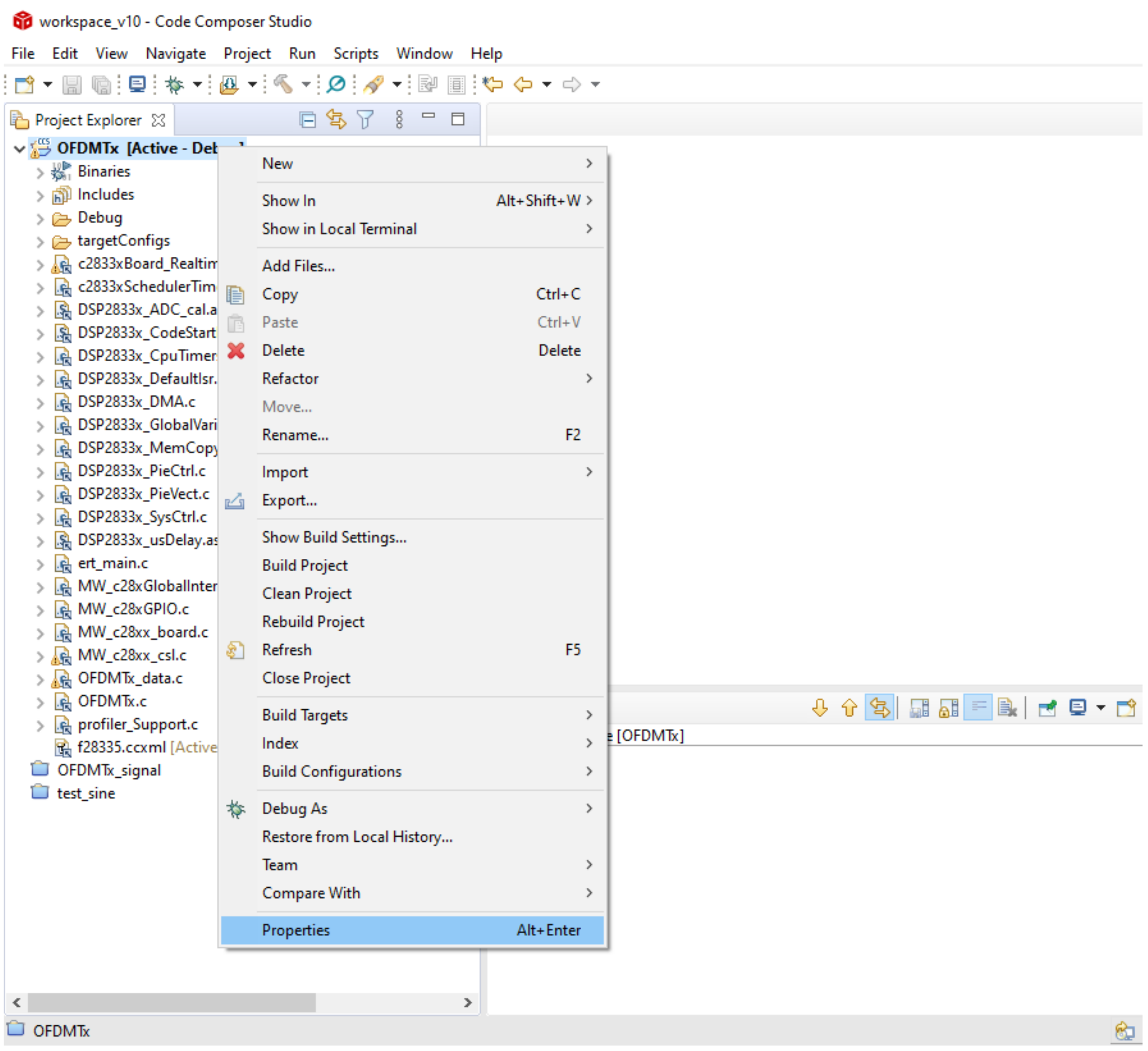
In addition to the files listed above there are a few more included in the zip file.
OFDMTx.slx – This is the model to be used in Simulink. It calls the function genOneFrame to get a data frame of size 160 samples and then sends it out through the GPIOs of the board one sample at a time. Notice that inside the "send to GPIO" Matlab function block the input value have been scaled to (input*0.4+0.5). This is a crude way to get by the sign bit issue. The proper scaling would depend on the driver that follows the congregated GPIOs.
genOneFrame.m – This function does nothing but calling generateOFDMSignal.
pltConstel.m – This function plots the constellation points before and after the frame equalization. Set a breakpoint inside OFDMReciever after calling of frameEqualization. When the breakpoint is reached, run pltConstel at the command prompt in Matlab workspace.
modified_demodulator.zip – This zip file contain files that have been partially optimized on the receiver side. These files includes: frameEqualization, mydemodulate, mygetDataLinearIndex, myOFDMDemodulator. The files are at a state where they are usable but not yet fully debugged.

Instructions to download and run the code using Code Composer Studio instead of using Simulink
1. Install Code composer Studio -- I am using version 10.2.0.00009.
2. Inside code composer, select File -> Open Projects from File System
3. At "Import source" select "Directory". And set it to the project directory generated by Simulink "CCS_Project". This folder is inside "OFDMTx_mbd_9_3/OFDMTx_ert_rtw".
4. Click "Finish", At this point the project is loaded and one can see the list of C files in the "Project Explorer" of the code composer studio, it is the window at the left margin.



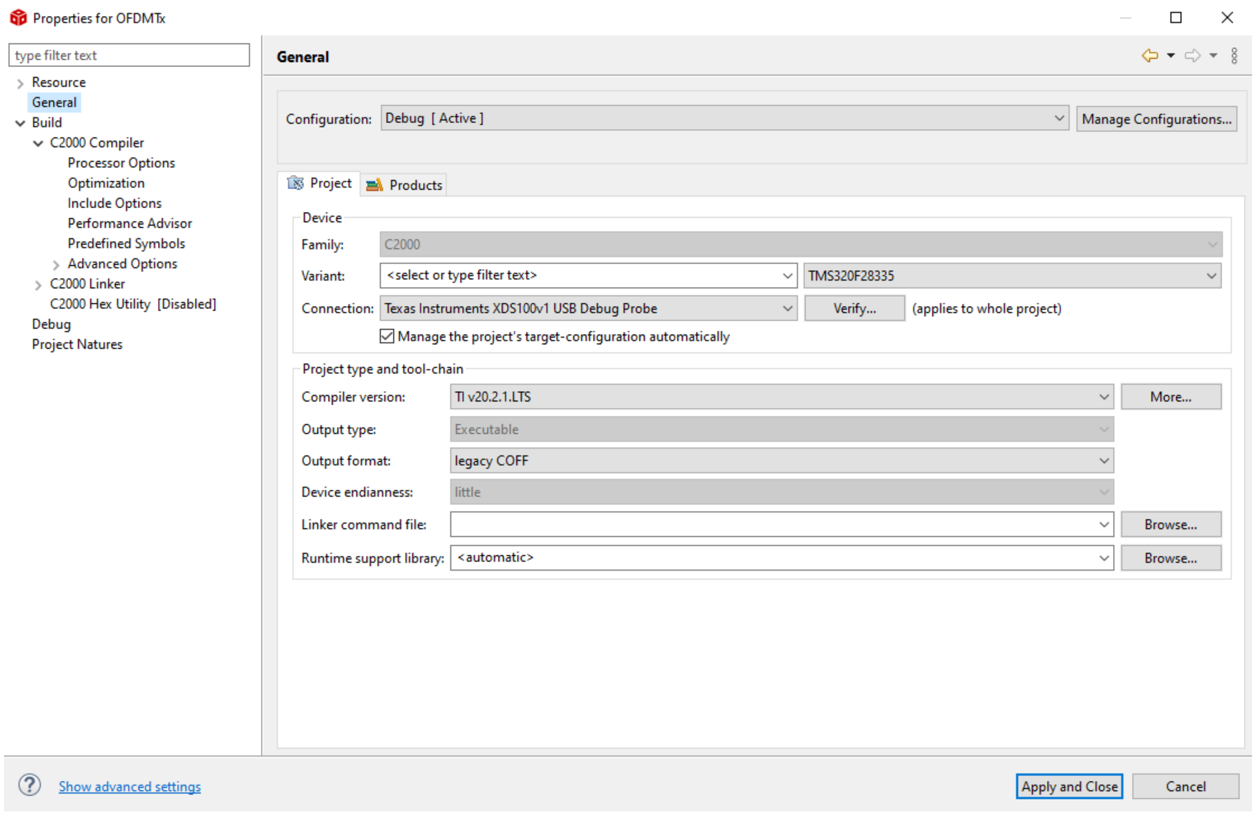
5. Assuming that at this point the board is connected to the PC.

6. Right click on the Project name “OFDMTx” and select “Properties”. One can do so also by typing “Alt-Enter”.



7. The "Properties for OFDMTx" window pop out. At the second selection on the left margin select "General". This is the general settings page.

8. At the general settings page, under the "project" tab, observe the "Variant:" row is set to "Custom C2000 Device". Change it to "TM320F28335". Please see the picture below.


Properties for OFDMTx

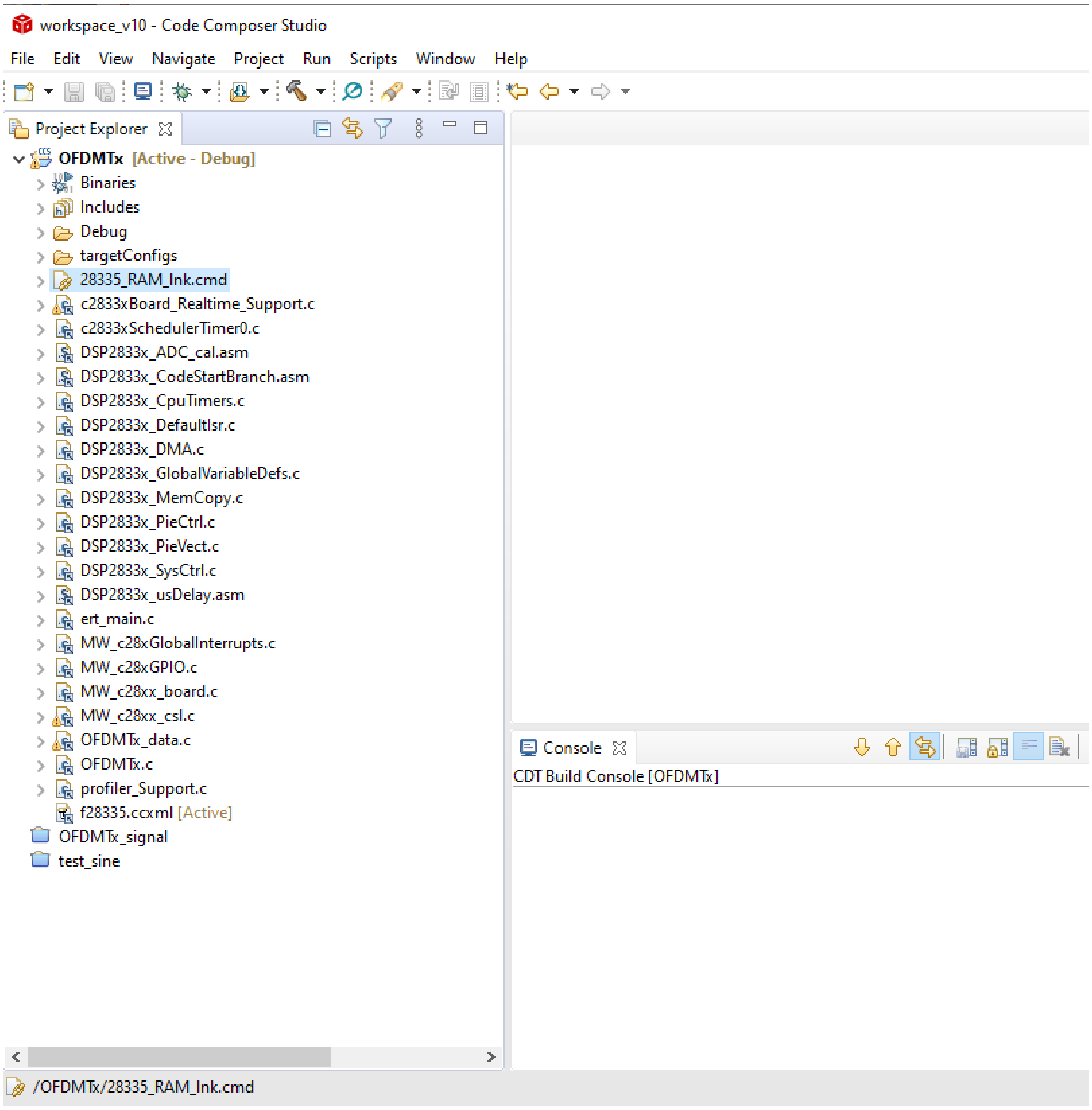
General

Configuration: Debug [Active]

Device
Family: C2000
Variant: <select or type filter text> TMS320F28335
Connection: Texas Instruments XDS100v1 USB Debug Probe (applies to whole project)
☑ Manage the project's target-configuration automatically

Project type and tool-chain
Compiler version: TI v20.2.1.LTS
Output type: Executable
Output format: legacy COFF
Device endianness: little
Linker command file:
Runtime support library: <automatic>

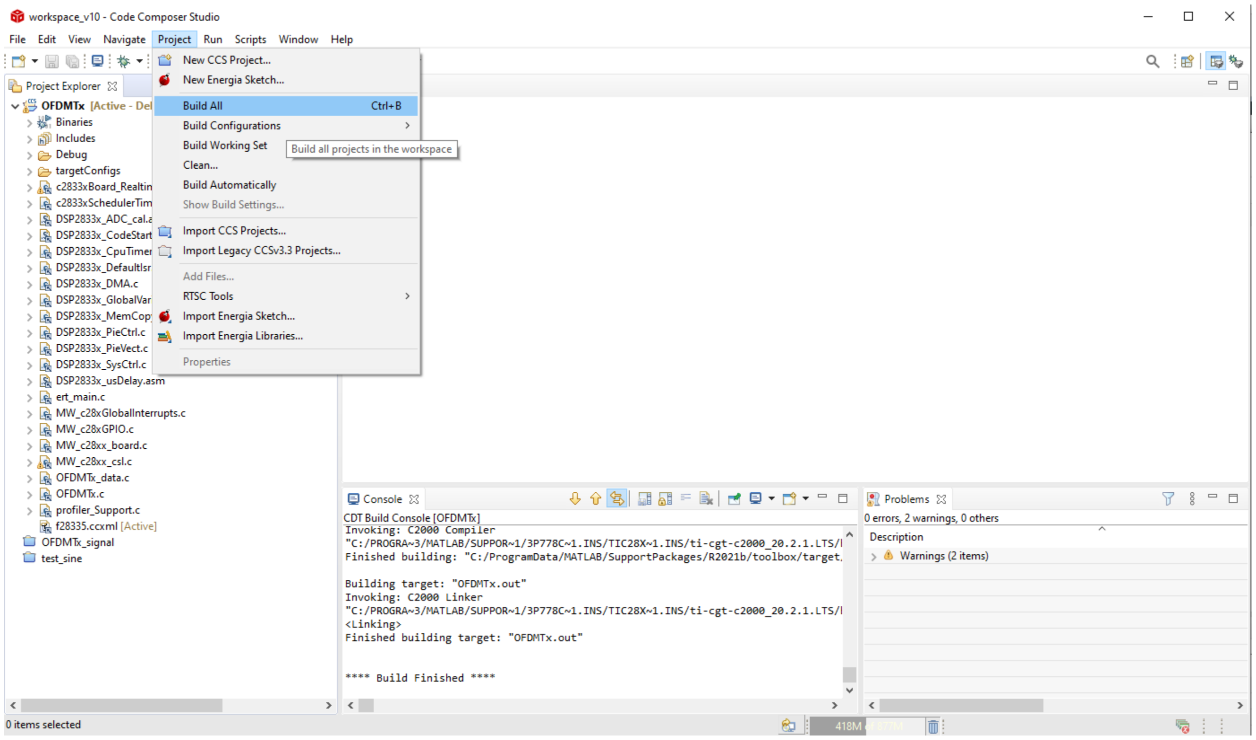
9. At the "Connection:" row, set it to "Texas Instruments XDS100v1 USB Debug Probe". You may also click "Verify" to see the confirmation message.
10. Select "Apply and Close", and you are now back to the workspace view. However, an extra file has now been added and need to be removed.
11. In the "Project Explorer" window, delete the file "28335_RAM_lnk.cmd" by right click and select delete.



OFDMTx [Active - Debug]
Binaries
Includes
Debug
targetConfigs
28335_RAM_lnk.cmd
c2833xBoard_Realtime_Support.c
c2833xSchedulerTimer0.c
DSP2833x_ADC_cal.asm
DSP2833x_CodeStartBranch.asm
DSP2833x_CpuTimers.c
DSP2833x_DefaultIsr.c
DSP2833x_DMA.c
DSP2833x_GlobalVariableDefs.c
DSP2833x_MemCopy.c
DSP2833x_PieCtrl.c
DSP2833x_PieVect.c
DSP2833x_SysCtrl.c
DSP2833x_usDelay.asm
ert_main.c
MW_c28xGlobalInterrupts.c
MW_c28xGPIO.c
MW_c28xx_board.c
MW_c28xx_csl.c
OFDMTx_data.c
OFDMTx.c
profiler_Support.c
f28335.ccxml [Active]
OFDMTx_signal
test_sine

/OFDMTx/28335_RAM_lnk.cmd

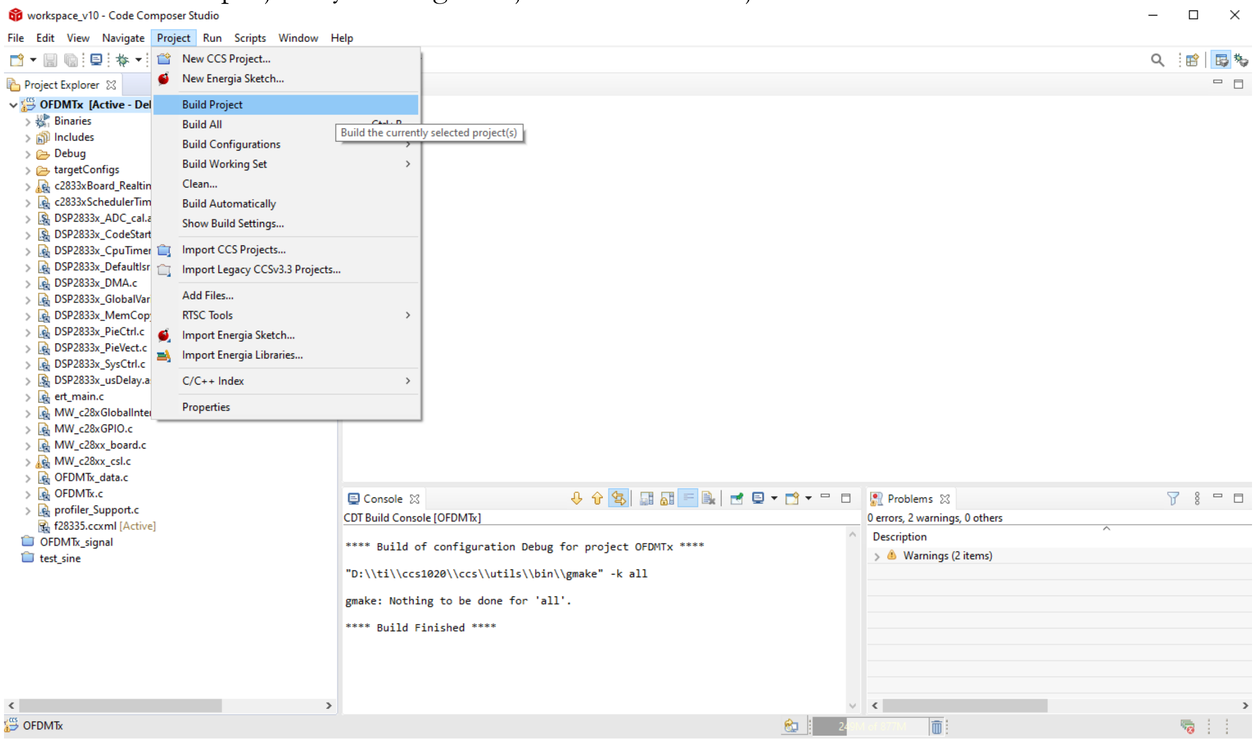
12. Click on “Project” -> “Build All”.



CDT Build Console [OFDMTx]

```
Invoking: C2000 Compiler
"C:/PROGRA~3/MATLAB/SUPPOR~1/3P778C~1.INS/TIC28X~1.INS/ti-cgt-c2000_20.2.1.LTS/
Finished building: "C:/ProgramData/MATLAB/SupportPackages/R2021b/toolbox/target.

Building target: "OFDMTx.out"
Invoking: C2000 Linker
"C:/PROGRA~3/MATLAB/SUPPOR~1/3P778C~1.INS/TIC28X~1.INS/ti-cgt-c2000_20.2.1.LTS/
<Linking>
Finished building target: "OFDMTx.out"


**** Build Finished ****
```

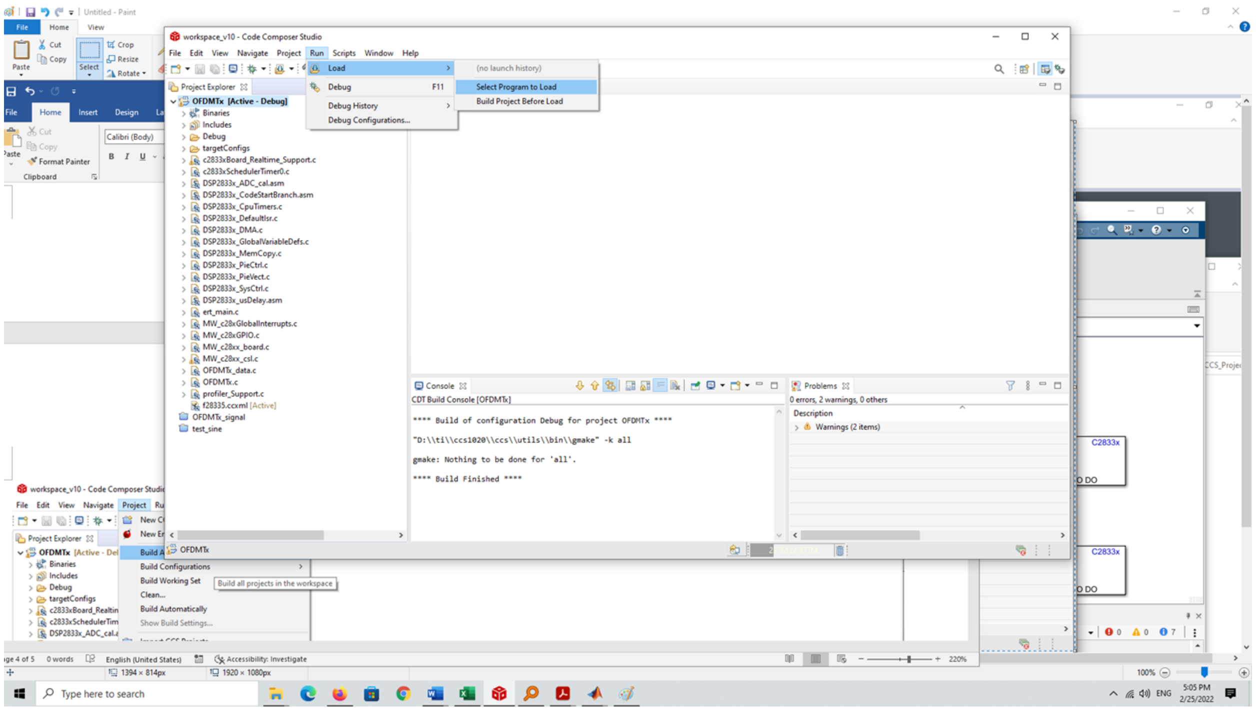
13. Build the project by clicking “Project”-> “Build Project”.



CDT Build Console [OFDMTx]

```
**** Build of configuration Debug for project OFDMTx ****

"D:\\ti\\ccs1020\\ccs\\utils\\bin\\gmake" -k all

gmake: Nothing to be done for 'all'.

**** Build Finished ****
```

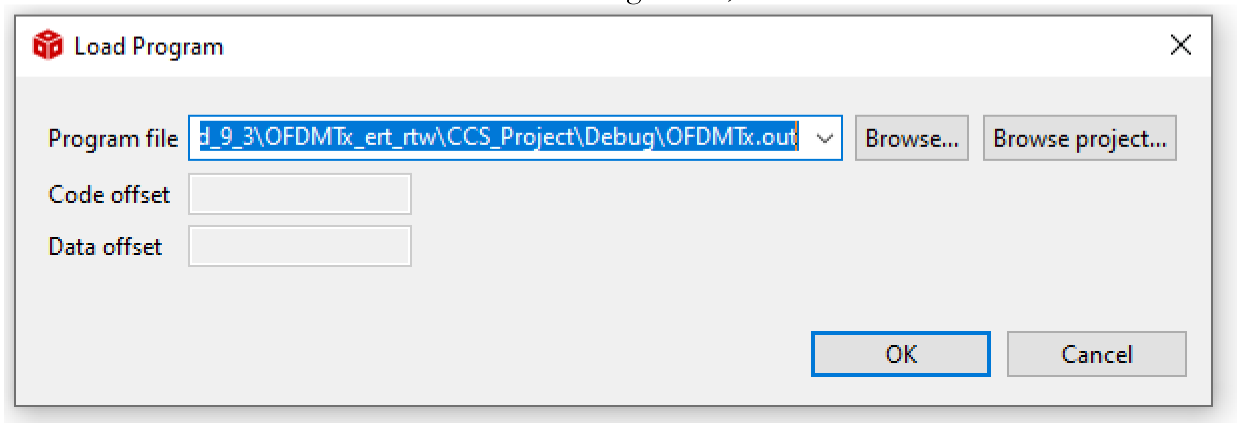
14. Click on “Run”-> “Load” -> “Select Program to Load”.



```
CDT Build Console [OFDMTx]
**** Build of configuration Debug for project OFDMTx ****
"D:\\ti\\ccs1020\\ccs\\utils\\bin\\gmake" -k all
gmake: Nothing to be done for 'all'.
**** Build Finished ****
```

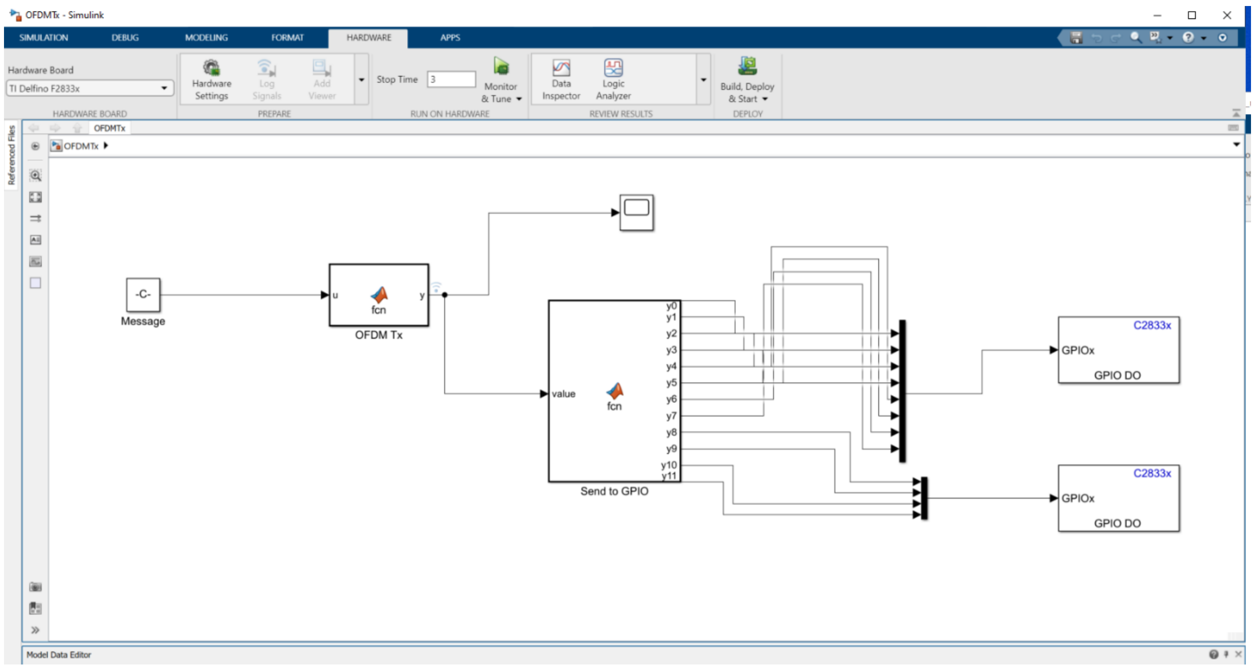
```
0 errors, 2 warnings, 0 others
Description
> Warnings (2 items)
```

15. Select the OFDMTx.out file. After clicking "OK", it will download and run on the board.



Program file	d_9_3\OFDMTx_ert_rtw\CCS_Project\Debug\OFDMTx.out
Code offset	
Data offset	

Instructions to modify the C code in Code Composer Studio
In version 10 (OFDMTx.slx) there are minor changes on the model code in Simulink. First, the GPIO outputs have been grouped together and be sent out in two blocks. Please see the picture below. Second, inside the block "OFDM Tx" calling of the data calculations "genOneFrame" has been placed at the end of the function. One can proceed to generate the C code for the TI chip using this model.



There are two changes that are needed to be modified in the generated C code. The first change would reduce the erroneous spikes lines generated due to the asynchronous setting of the GPIOs. The second change would allow the code to be run at a much higher speed by moving the calculation of the data outside the interrupt service.

Setting the GPIOs efficiently

In OFDMTx.c, comment out setting of each GPIO bit and replace it with setting all the bits together in one line. The resulting code is shown below:

```
uint32_T tmp; // extend this word to 32 bits

/* SignalConversion generated from: '<Root>/Digital Output1' incorporates:
     *  MATLAB Function: '<Root>/Send to GPIO'
     *  SignalConversion generated from: '<Root>/Digital Output'
     */
    GpioDataRegs.GPADAT.all = tmp;   // this will set all the GPIOs together

// All the below codes are commented out
#if 0
  OFDMTx_B.TmpSignalConversionAtDigitalOut[0] = ((tmp & 1U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[1] = ((tmp & 2U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[2] = ((tmp & 4U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[3] = ((tmp & 8U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[4] = ((tmp & 16U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[5] = ((tmp & 32U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[6] = ((tmp & 64U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[7] = ((tmp & 128U) != 0U);

  /* S-Function (c280xgpio_do): '<Root>/Digital Output' */
  {
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[0])
      GpioDataRegs.GPASET.bit.GPIO0 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO0 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[1])
```

```
      GpioDataRegs.GPASET.bit.GPIO1 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO1 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[2])
      GpioDataRegs.GPASET.bit.GPIO2 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO2 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[3])
      GpioDataRegs.GPASET.bit.GPIO3 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO3 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[4])
      GpioDataRegs.GPASET.bit.GPIO4 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[5])
      GpioDataRegs.GPASET.bit.GPIO5 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[6])
      GpioDataRegs.GPASET.bit.GPIO6 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[7])
      GpioDataRegs.GPASET.bit.GPIO7 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;
  }

  /* SignalConversion generated from: '<Root>/Digital Output1' incorporates:
   *  MATLAB Function: '<Root>/Send to GPIO'
   */
  OFDMTx_B.TmpSignalConversionAtDigitalOut[0] = ((tmp & 256U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[1] = ((tmp & 512U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[2] = ((tmp & 1024U) != 0U);
  OFDMTx_B.TmpSignalConversionAtDigitalOut[3] = ((tmp & 2048U) != 0U);

  /* S-Function (c280xgpio_do): '<Root>/Digital Output1' */
  {
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[0])
      GpioDataRegs.GPASET.bit.GPIO8 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[1])
      GpioDataRegs.GPASET.bit.GPIO9 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[2])
      GpioDataRegs.GPASET.bit.GPIO10 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
    if (OFDMTx_B.TmpSignalConversionAtDigitalOut[3])
      GpioDataRegs.GPASET.bit.GPIO11 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
  }
#endif
```

A.5.1. *Moving data calculations outside the interrupt service*

There are two main parts to this change. This first part is to extract part of the function OFDMTx_step inside OFDMTx.c into a new function OFDMTx_output. The second part is to modify the main loop in ert_main.c to run OFDMTx_step and running OFDMTx_output inside rt_OneStep.

1, Create a new function OFDMTx_output. The content of this function is extracted from OFDMTx_step.

```
void OFDMTx_output(void)
{
    real_T rtb_y;
    real_T v;
    uint32_T tmp;

    /* MATLAB Function: '<Root>/OFDM Tx' incorporates:
     *  Constant: '<Root>/Message'
     */
    if (OFDMTx_DW.timeCount == 1.0) {
      OFDMTx_DW.genFlag = true;
      OFDMTx_DW.outTogFlag = !OFDMTx_DW.outTogFlag;
      OFDMTx_DW.frameCount++;
      if (OFDMTx_DW.frameCount > 15.0) {
        OFDMTx_DW.frameCount = 1.0;
      }
    }

    OFDMTx_DW.timeCount++;
    if (OFDMTx_DW.timeCount > 160.0) {
      OFDMTx_DW.timeCount = 1.0;
    }

    if (OFDMTx_DW.outTogFlag) {
      rtb_y = OFDMTx_DW.outputBuf1[(int16_T)OFDMTx_DW.timeCount - 1];
    } else {
      rtb_y = OFDMTx_DW.outputBuf2[(int16_T)OFDMTx_DW.timeCount - 1];
    }

    /* MATLAB Function: '<Root>/Send to GPIO' */
    rtb_y = (rtb_y * 0.2 + 0.2) * 4096.0;
    v = fabs(rtb_y);
    if (v < 4.503599627370496E+15) {
      if (v >= 0.5) {
        rtb_y = floor(rtb_y + 0.5);
      } else {
        rtb_y *= 0.0;
      }
    }

    if (rtb_y < 4096.0) {
      if (rtb_y >= 0.0) {
        tmp = (uint16_T)rtb_y;
      } else {
        tmp = 0U;
      }
    } else {
      tmp = 4095U;
    }

    /* SignalConversion generated from: '<Root>/Digital Output1' incorporates:
     *  MATLAB Function: '<Root>/Send to GPIO'
     *  SignalConversion generated from: '<Root>/Digital Output'
     */
    GpioDataRegs.GPADAT.all = tmp;


}
void OFDMTx_step(void)
{

  if (OFDMTx_DW.genFlag) {
    if (!OFDMTx_DW.outTogFlag) {
      if (!OFDMTx_DW.OFDMTX_not_empty) {
        OFDMTx_DW.OFDMTX.isInitialized = 0L;
        memcpy(&OFDMTx_DW.OFDMTX.pPreamble[0], &OFDMTx_P.params.pPreamble[0],
               640U * sizeof(real_T));
        memcpy(&OFDMTx_DW.OFDMTX.pPilots[0], &OFDMTx_P.params.pPilots[0], 44U *
               sizeof(real_T));
        OFDMTx_DW.OFDMTX.matlabCodegenIsDeleted = false;
        OFDMTx_DW.OFDMTX_not_empty = true;
      }

      OFDMTx_SystemCore_step(&OFDMTx_DW.OFDMTX, OFDMTx_P.messageBinary,
        OFDMTx_DW.outputBuf1);
    } else {
      if (!OFDMTx_DW.OFDMTX_not_empty) {
        OFDMTx_DW.OFDMTX.isInitialized = 0L;
        memcpy(&OFDMTx_DW.OFDMTX.pPreamble[0], &OFDMTx_P.params.pPreamble[0],
               640U * sizeof(real_T));
        memcpy(&OFDMTx_DW.OFDMTX.pPilots[0], &OFDMTx_P.params.pPilots[0], 44U *
               sizeof(real_T));
        OFDMTx_DW.OFDMTX.matlabCodegenIsDeleted = false;
        OFDMTx_DW.OFDMTX_not_empty = true;
      }

      OFDMTx_SystemCore_step(&OFDMTx_DW.OFDMTX, OFDMTx_P.messageBinary,
        OFDMTx_DW.outputBuf2);
    }

    OFDMTx_DW.genFlag = false;
  }

}
```

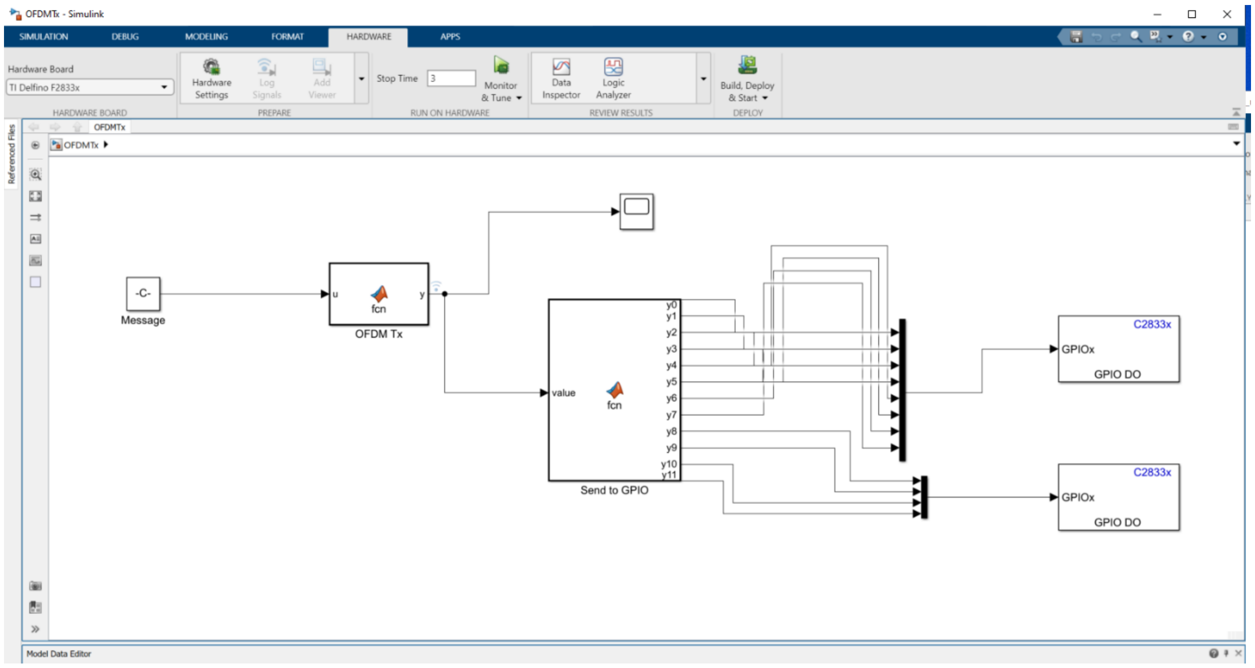
2, Add the new function OFDMTx_output into the header file OFDMTx.h.

```
extern void OFDMTx_step(void);
extern void OFDMTx_output(void);
```

3, Inside ert_main.c, calling OFDMTx_output inside rt_OneStep and calling OFDMTx_step inside the main function under the while(runModel).

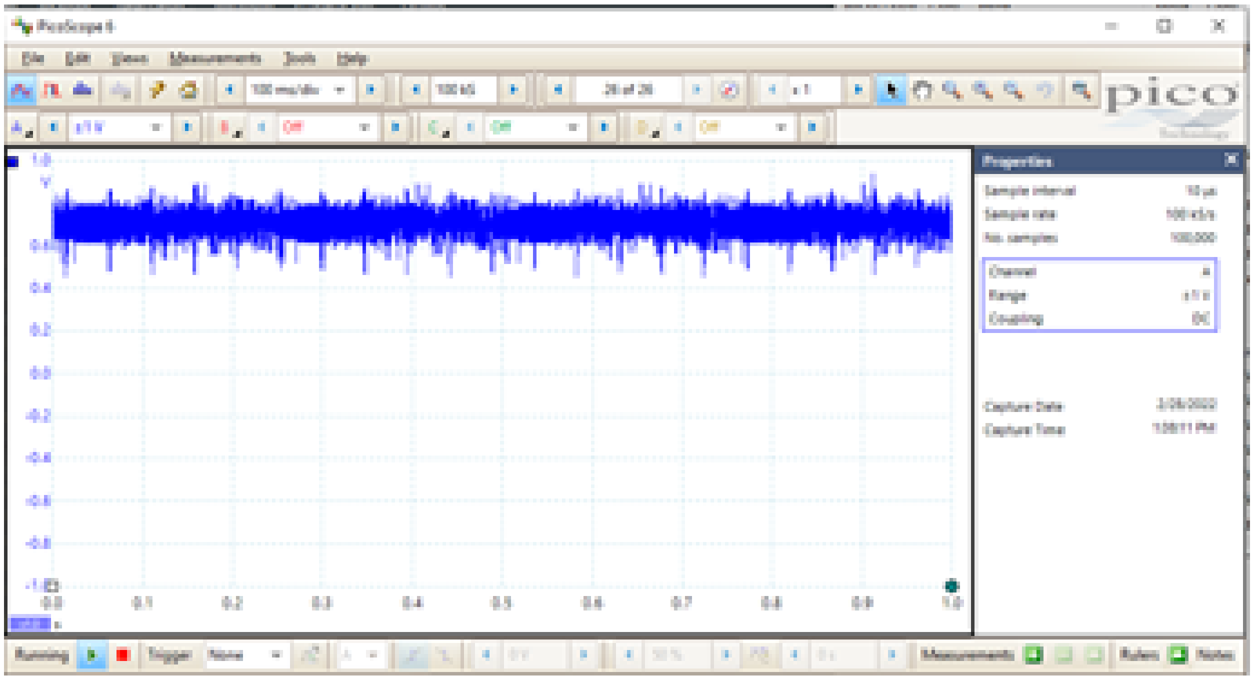
```
void rt_OneStep(void)
{
  /* Check for overrun. Protect OverrunFlag against preemption */
  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }

//  enableTimer0Interrupt();
  OFDMTx_output();

  /* Get model outputs here */
//  disableTimer0Interrupt();
  OverrunFlag--;
}

Int main(void)
{

   while (runModel) {

      OFDMTx_step();

      stopRequested = !( rtmGetErrorStatus(OFDMTx_M) == (NULL));
  }
}
```

Instructions to sample data from Picoscope and to recover the message
1, Set the sampling frequency to a value which the scope can support. The sampling frequency can be set inside the Simulink model or in the generated C code. Inside the "Message" block of the model, one can set the proper sampling frequency. The "Sample time" field of the block should be set to 1/sampling Frequency.

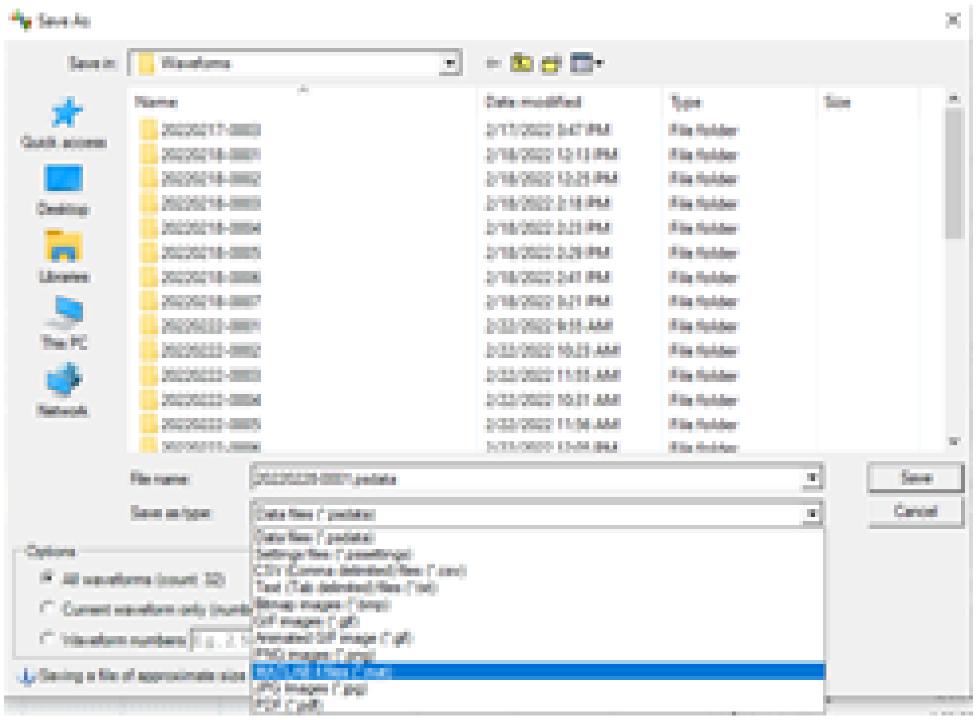


The sample time can also be changed inside the file ert_main.c, the first line in the main function, float modelBaseRate = 0.0001;

2, On the PicoScope 6 application, adjust the time per division and frame size until the target sample rate is achieved. The sample rate is displayed on the right margin under "properties".



3, To save the captured samples into a file, click on "File" -> "Save As". At the window that popped out select the "Save as type" as "MATLAB 4 files (*.mat)".


Save As

Save in: Waveforms

Name	Date modified	Type	Size
20220217-0003	2/17/2022 3:47 PM	File folder	
20220218-0001	2/18/2022 12:13 PM	File folder	
20220218-0002	2/18/2022 12:25 PM	File folder	
20220218-0003	2/18/2022 2:18 PM	File folder	
20220218-0004	2/18/2022 2:23 PM	File folder	
20220218-0005	2/18/2022 2:29 PM	File folder	
20220218-0006	2/18/2022 2:41 PM	File folder	
20220218-0007	2/18/2022 3:21 PM	File folder	
20220222-0001	2/22/2022 9:55 AM	File folder	
20220222-0002	2/22/2022 10:23 AM	File folder	
20220222-0003	2/22/2022 11:55 AM	File folder	
20220222-0004	2/22/2022 10:21 AM	File folder	
20220222-0005	2/22/2022 11:56 AM	File folder	

File name: 20220229-0001.psdata

Save as type: Data files (*.psdata)

Data files (*.psdata)
Settings files (*.pssettings)
CSV (Comma delimited) files (*.csv)
Text (Tab delimited) files (*.txt)
Bitmap images (*.bmp)
GIF images (*.gif)
Animated GIF image (*.gif)
PNG images (*.png)
MATLAB 4 files (*.mat)
JPG images (*.jpg)
PDF (*.pdf)

Options
- [✓ All waveforms (count: 32)]
- Current waveform only
- Waveform numbers

Saving a file of approximate size

Save
Cancel

4, Modify prepData.m to have the variable "label" set to the folder of the latest "mat" file saved. Run prepData.m followed by "justDoReceive.m". The received constellation and the demodulated message would be displayed.

A.6. Data Link Software

A.6.1. Main C code

```
/*
 * File: ert_main.c
 *
 * Code generated for Simulink model 'OFDMTx'.
 *
 * Model version                  : 2.109
 * Simulink Coder version         : 9.6 (R2021b) 14-May-2021
 * C/C++ source code generated on : Tue Feb 22 10:15:15 2022
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Texas Instruments->C2000
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include "OFDMTx.h"
#include "rtwtypes.h"

volatile int IsrOverrun = 0;
static boolean_T OverrunFlag = 0;
void rt_OneStep(void)
{
  /* Check for overrun. Protect OverrunFlag against preemption */
  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }

//  enableTimer0Interrupt();
  OFDMTx_output();

  /* Get model outputs here */
//  disableTimer0Interrupt();
  OverrunFlag--;
}

volatile boolean_T stopRequested;
volatile boolean_T runModel;
int main(void)
{
  float modelBaseRate = 0.00001;
  float systemClock = 100;

  /* Initialize variables */
  stopRequested = false;
  runModel = false;
  c2000_flash_init();
  init_board();

#ifdef MW_EXEC_PROFILER_ON

  config_profilerTimer();

#endif

  ;
  rtmSetErrorStatus(OFDMTx_M, 0);
  OFDMTx_initialize();
  globalInterruptDisable();
  configureTimer0(modelBaseRate, systemClock);
  runModel =
    rtmGetErrorStatus(OFDMTx_M) == (NULL);
  enableTimer0Interrupt();
  globalInterruptEnable();
  while (runModel) {

      OFDMTx_step();

    stopRequested = !(
                      rtmGetErrorStatus(OFDMTx_M) == (NULL));
  }

  /* Terminate model */
  OFDMTx_terminate();
  globalInterruptDisable();
  return 0;
}
```

```
/*
 * File trailer for generated code.
 *
 * [EOF]
 */
```

A.6.2. SM320F28335-HT Clock Register Settings

```
// TI File Revision: /main/8
// Checkin Date: October 23, 2007   11:29:25
//###########################################################################
//
// FILE:   DSP2833x_Examples.h
//
// TITLE:  DSP2833x Device Definitions.
//
//###########################################################################
// $TI Release: DSP2833x Header Files V1.10 $
// $Release Date: February 15, 2008 $
//###########################################################################

#ifndef DSP2833x_EXAMPLES_H
#define DSP2833x_EXAMPLES_H


#ifdef __cplusplus
extern "C" {
#endif


/*-----------------------------------------------------------------------------
      Specify the PLL control register (PLLCR) and divide select (DIVSEL) value.
-----------------------------------------------------------------------------*/
//#define DSP28_DIVSEL   0   // Enable /4 for SYSCLKOUT
//#define DSP28_DIVSEL   1 // Enable /4 for SYSCKOUT
#define DSP28_DIVSEL     2 // Enable /2 for SYSCLKOUT
//#define DSP28_DIVSEL     3 // Enable /1 for SYSCLKOUT

#define DSP28_PLLCR   10
//#define DSP28_PLLCR    9
//#define DSP28_PLLCR    8
//#define DSP28_PLLCR    7
//#define DSP28_PLLCR    6
//#define DSP28_PLLCR    5
//#define DSP28_PLLCR    4
//#define DSP28_PLLCR    3
//#define DSP28_PLLCR    2
//#define DSP28_PLLCR    1
//#define DSP28_PLLCR    0  // PLL is bypassed in this mode
//----------------------------------------------------------------------------


/*-----------------------------------------------------------------------------
```

```
      Specify the clock rate of the CPU (SYSCLKOUT) in nS.

      Take into account the input clock frequency and the PLL multiplier
      selected in step 1.

      Use one of the values provided, or define your own.
      The trailing L is required tells the compiler to treat
      the number as a 64-bit value.

      Only one statement should be uncommented.

      Example 1:150 MHz devices:
                CLKIN is a 30MHz crystal.

                In step 1 the user specified PLLCR = 0xA for a
                150Mhz CPU clock (SYSCLKOUT = 150MHz).

                In this case, the CPU_RATE will be 6.667L
                Uncomment the line:  #define CPU_RATE  6.667L

      Example 2:  100 MHz devices:
                  CLKIN is a 20MHz crystal.

                     In step 1 the user specified PLLCR = 0xA for a
                     100Mhz CPU clock (SYSCLKOUT = 100MHz).

                     In this case, the CPU_RATE will be 10.000L
                  Uncomment the line:  #define CPU_RATE  10.000L
-----------------------------------------------------------------------------*/
//#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    7.143L   // for a 140MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    8.333L   // for a 120MHz CPU clock speed (SYSCLKOUT)
#define CPU_RATE   10.000L   // for a 100MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   13.330L   // for a 75MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   33.333L   // for a 30MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   41.667L   // for a 24MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   50.000L   // for a 20MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   66.667L   // for a 15MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE  100.000L   // for a 10MHz CPU clock speed  (SYSCLKOUT)

//----------------------------------------------------------------------------

/*-----------------------------------------------------------------------------
      Target device (in DSP2833x_Device.h) determines CPU frequency
      (for examples) - either 150 MHz (for 28335 and 28334) or 100 MHz
      (for 28332). User does not have to change anything here.
-----------------------------------------------------------------------------*/
#if DSP28_28332                   // DSP28_28332 device only
  #define CPU_FRQ_100MHZ    1     // 100 Mhz CPU Freq (20 MHz input freq)
  #define CPU_FRQ_150MHZ    0
#else
  #define CPU_FRQ_100MHZ    1     // DSP28_28335||DSP28_28334
  #define CPU_FRQ_150MHZ    0     // 150 MHz CPU Freq (30 MHz input freq) by DEFAULT
#endif
```

```
//---------------------------------------------------------------------------
// Include Example Header Files:
//

#include "DSP2833x_GlobalPrototypes.h"         // Prototypes for global functions
within the
                                              // .c files.

#include "DSP2833x_EPwm_defines.h"             // Macros used for PWM examples.
#include "DSP2833x_Dma_defines.h"              // Macros used for DMA examples.
#include "DSP2833x_I2c_defines.h"              // Macros used for I2C examples.

#define PARTNO_28335  0xFA
#define PARTNO_28334  0xF9
#define PARTNO_28332  0xF8


// Include files not used with DSP/BIOS
#ifndef DSP28_BIOS
#include "DSP2833x_DefaultISR.h"
#endif


// DO NOT MODIFY THIS LINE.
#define DELAY_US(A)  DSP28x_usDelay(((((long double) A * 1000.0L) / (long
double)CPU_RATE) - 9.0L) / 5.0L)


#ifdef __cplusplus
}
#endif /* extern "C" */

#endif  // end of DSP2833x_EXAMPLES_H definition


//===========================================================================
// End of file.
//===========================================================================
```

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Avery Cashion	05965	atcashi@sandia.gov
Giorgia Bettin	08916	gbettin@sandia.gov
Andrew Wright	08916	aawrigh@sandia.gov
Douglas Blankenship	08910	dablank@sandia.gov
Technical Library	1911	sanddocs@sandia.gov

Email—External

Name	Company Email Address	Company Name
Francis Tiong	ftiong@mathworks.com	MathWorks
Zachary Frone	zachary.frone@ee.doe.gov	DOE
Lauren Boyd	lauren.boyd@ee.doe.gov	DOE

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
5	Andrew A. Wright	08916	MS 1033

Hardcopy—External

Number of Copies	Name	Company Name and Company Mailing Address
1	Zachary Frone	DOE GTO
1	Lauren Boyd	DOE GTO

This page left blank