cupl Tag Documentation

Release 0.0.1

Malcolm Mackay

Feb 04, 2022

FIRMWARE

CHAPTER

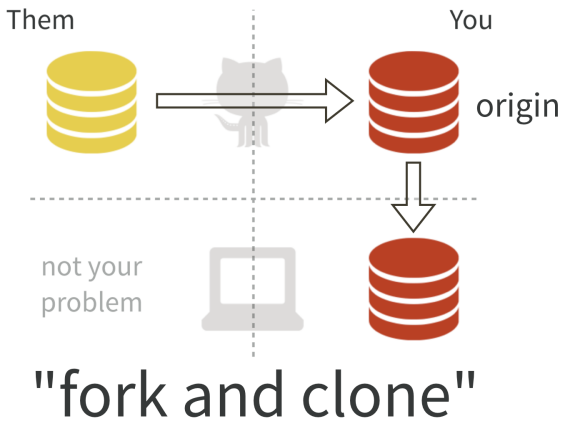
ONE

GETTING STARTED

1.1 Prerequisites

- MSP-FET debugger (TI).
- GitHub Desktop (download) or your choice of Git software.

1.2 Fork and Clone cupl Tag

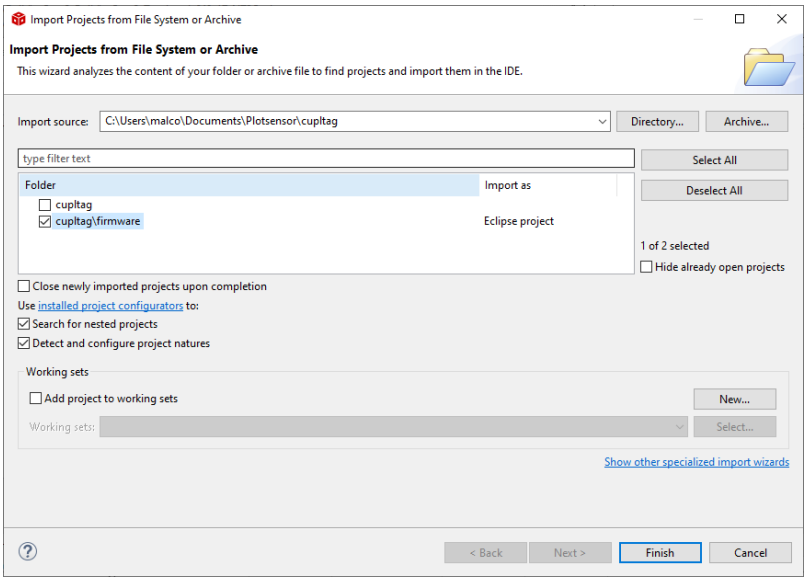


Visit the cupl Tag repository on (GitHub). Click the fork button in the top right.

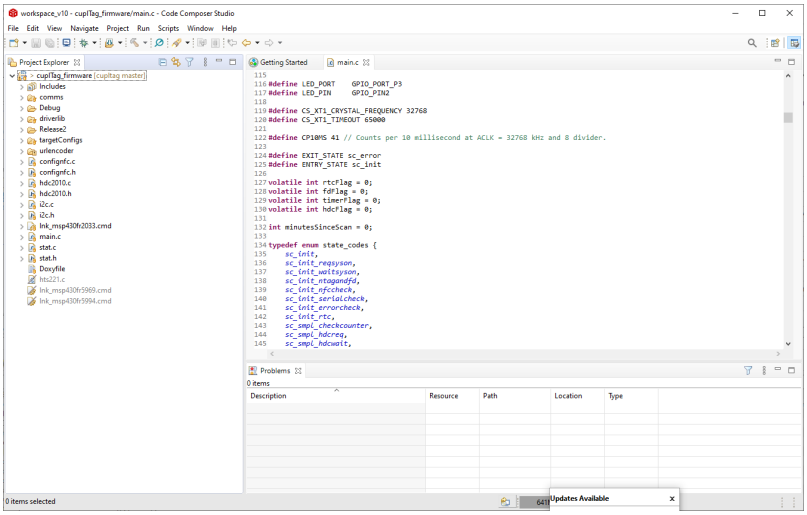
Clone the forked repository to your computer by clicking the green Clone or Download button. See https://guides.github.com/activities/forking/ for more details.

1.3 Open the Code Composer Project

1. Download and install Code Composer Studio 10 (CCS).
2. Open CCS. Launch the workspace of your choice.
3. Click File -> Open Projects from File System...
4. In Import Source, select the folder containing the clone of cupl Tag.



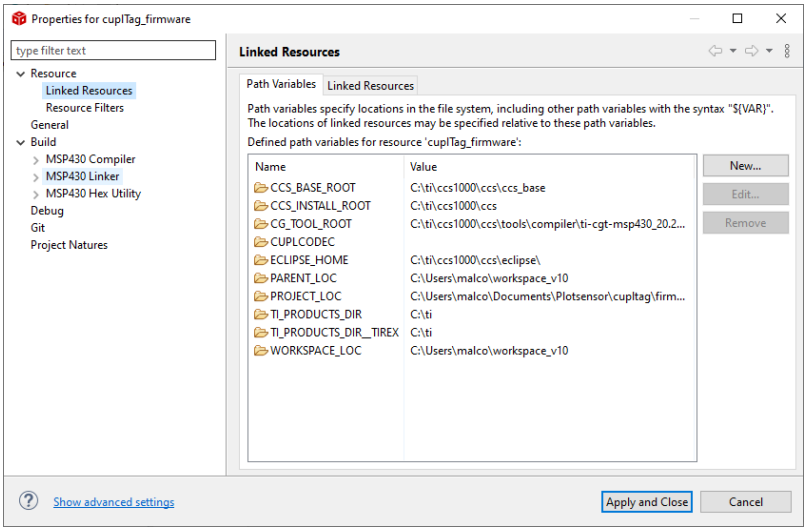
5. Ensure that cupltag\firmware is **checked**. Every other folder should be unchecked.
6. Click Finish.
7. The project is now open.



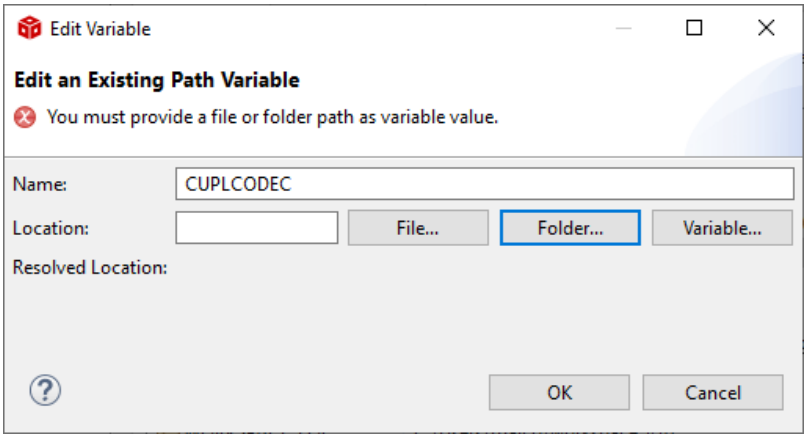
1.4 Add Reference to cuplCodec

The cuplTag firmware depends on C files from cuplCodec.

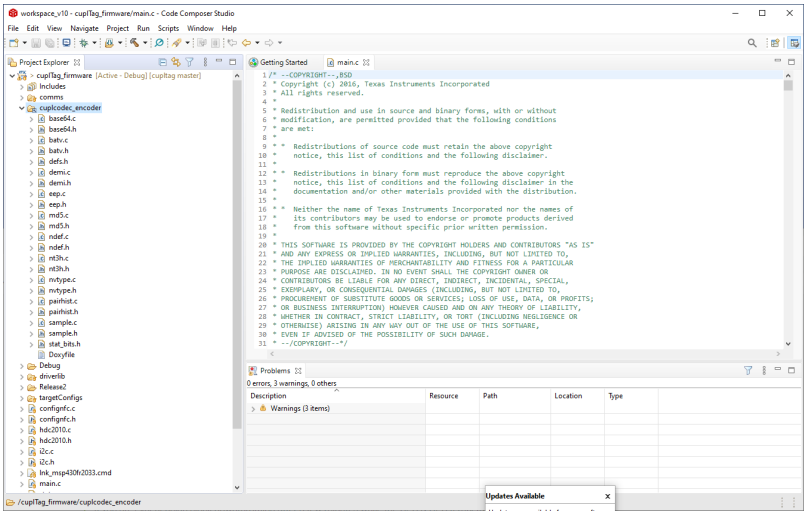
1. Fork the cuplCodec repository on (GitHub).
2. Clone this into a folder on your computer.
3. In CCS, right click the cuplTag_firmware project.
4. Select Properties from the context menu.
5. In the Properties window, expand Resource on the left hand panel and select Linked Resources.



6. Double click the CUPLCODEC path variable. The Edit Variable window will appear.



7. Click the Folder… button. Select the Codec clone folder from step 2.
8. Click Apply and Close.
9. The cuplcodec_encoder project folder will now contain references to files inside cupl Codec.



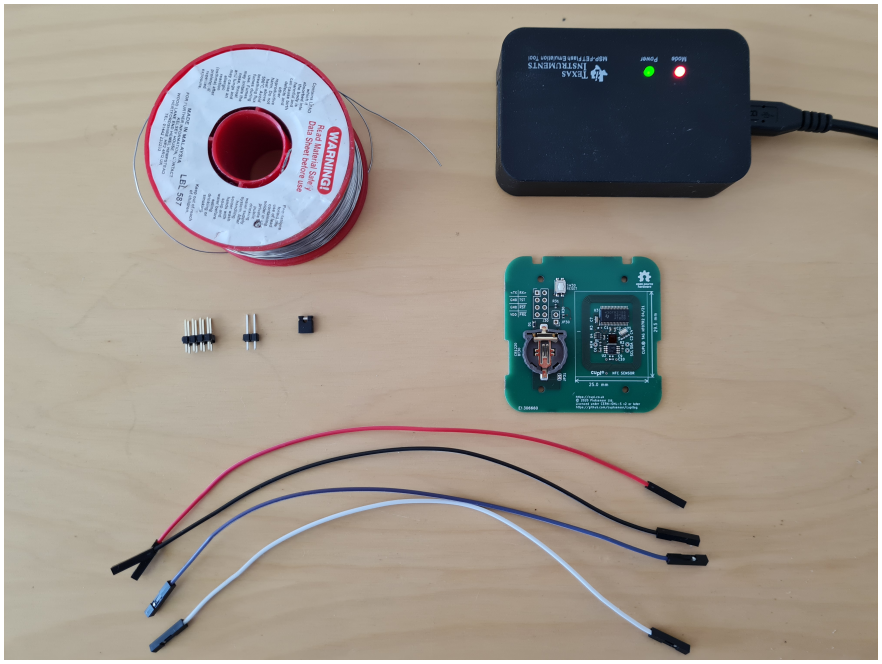
CHAPTER

TWO

PROGRAMMING

These instructions demonstrate how to program and debug the MSP430 on cuplTag.

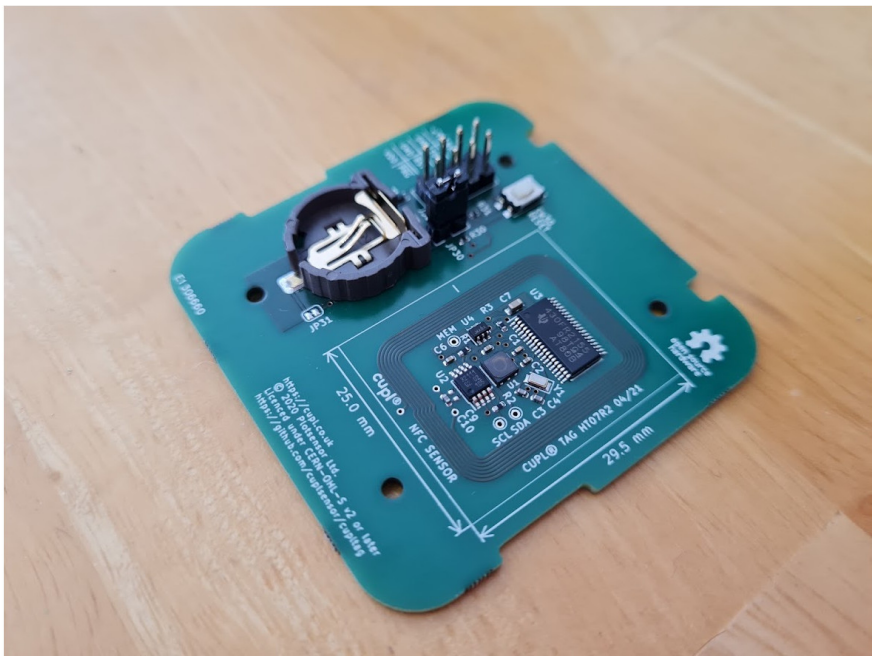
2.1 Equipment



You will need:

- An MSP-FET with a USB cable.
- A PC running Code Composer Studio.
- 4 coloured jumper wires.
- A 2x4 way 2.54mm pitch pin header.
- A 1x2 way 2.54mm pitch pin header.
- A 2 way jumper.
- Solder.
- A cuplTag PCBA (HT07), unscrewed from the enclosure, with no battery inserted.

2.2 Populate the Headers



First, solder the pin headers onto J30 and JP30 of HT07. Use the jumper to short JP30.

2.3 Make Connections

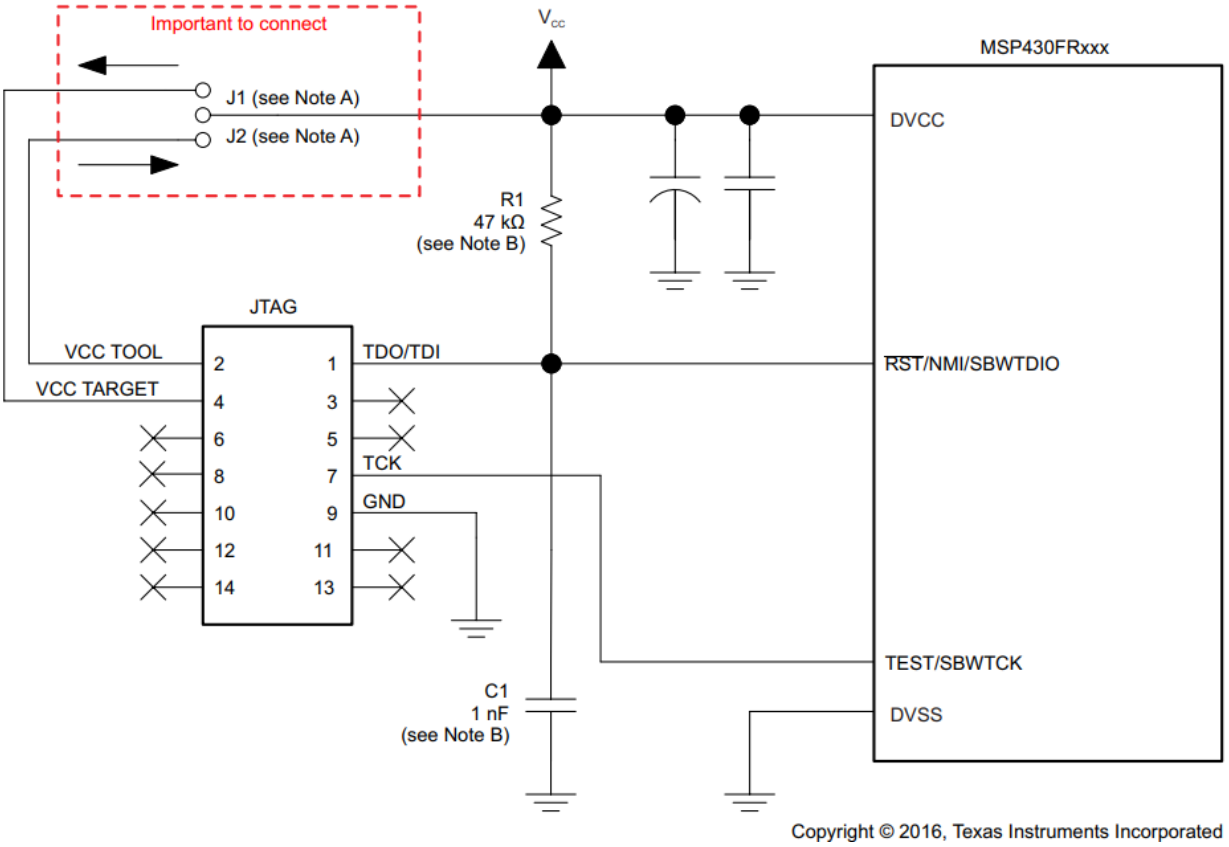
In the diagram below we will make connection J2 instead of J1, because the HT07 has no battery inserted.

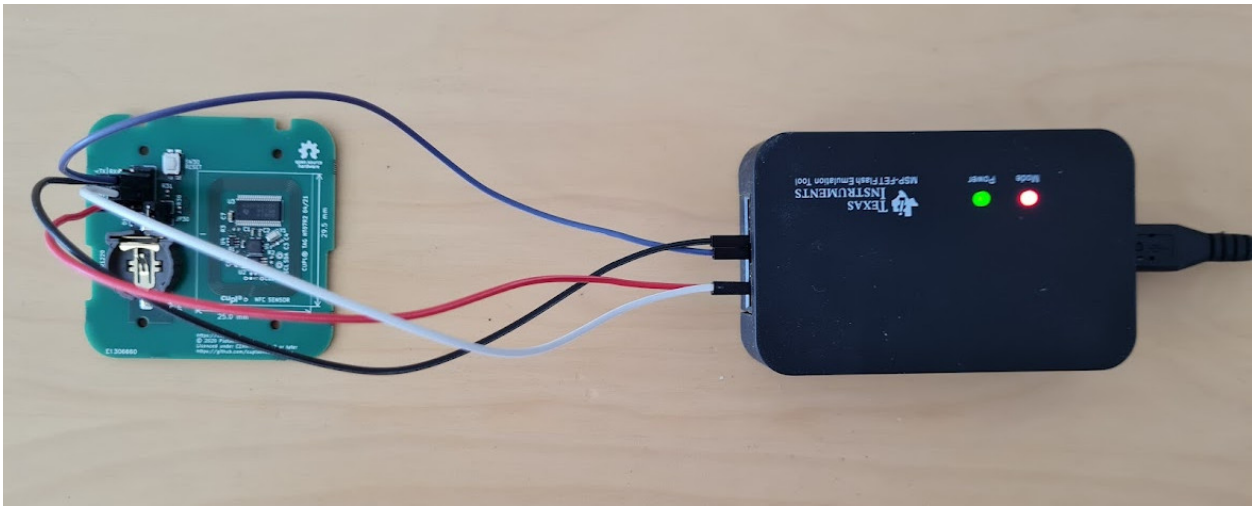
Spy-Bi-Wire is used to progra . Connect it to the MSP-FET.

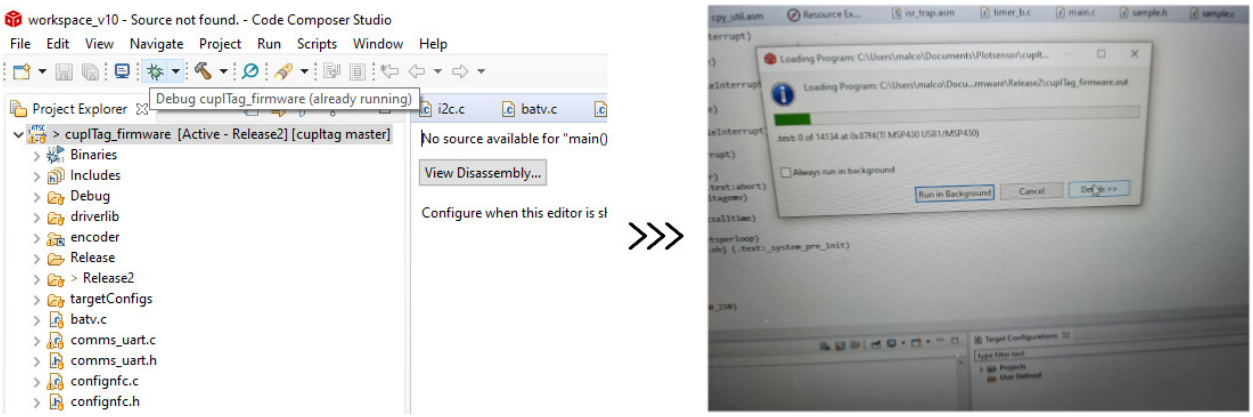
Name	Colour	MSP-FET name	MSP-FET pin	HT07 J30 pin	HT07 J30 name
+3V3	Red	VCC_TOOL	2	7	VDD
GND	Black	GND	9	3	GND
SBWTDIO	White	TDO/TDI	1	6	nRST
SBWTCK	Purple	TCK	7	4	TST

2.4 Program in CCS

1. Connect the MSP-FET to a PC with a USB cable.
2. Open the Code Composer Studio cuplTag project *created earlier <GettingStarted>*.
3. Click on the Debug button. Wait for programming to complete.


Important to connect
J1 (see Note A)
J2 (see Note A)
V_{CC}
MSP430FRxxx
DVCC
R1
47 kΩ
(see Note B)
JTAG
VCC TOOL
VCC TARGET
TDO/TDI
RST/NMI/SBWTDIO
2
1
4
3
6
5
8
7
10
9
12
11
14
13
TCK
GND
TEST/SBWTCK
C1
1 nF
(see Note B)
DVSS
Copyright © 2016, Texas Instruments Incorporated

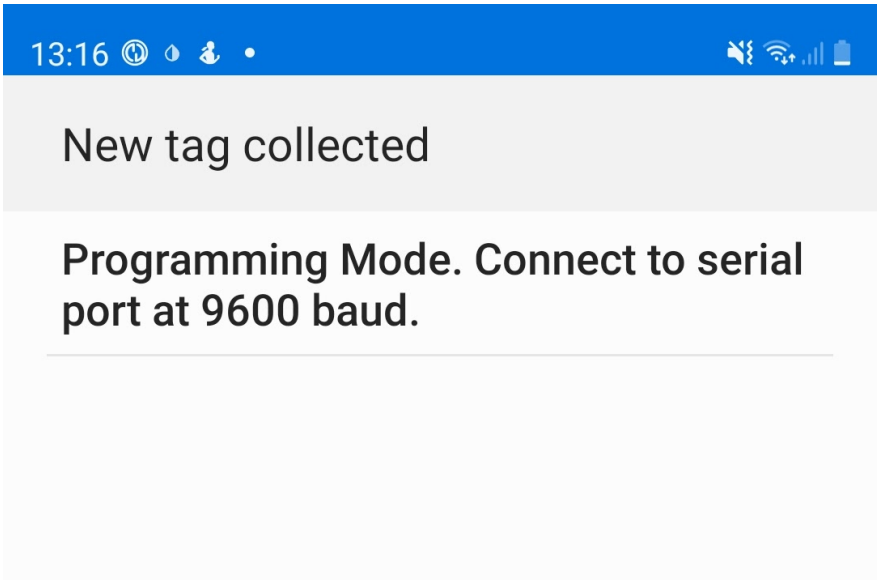




2.5 Test

Test the program has loaded correctly by scanning HT07 with your phone.

If JP30 is shorted, the MSP430 will boot into *programming mode*: The serial port is enabled and a status string is written to an NDEF text record on the tag.



CHAPTER

THREE

STATE CHART

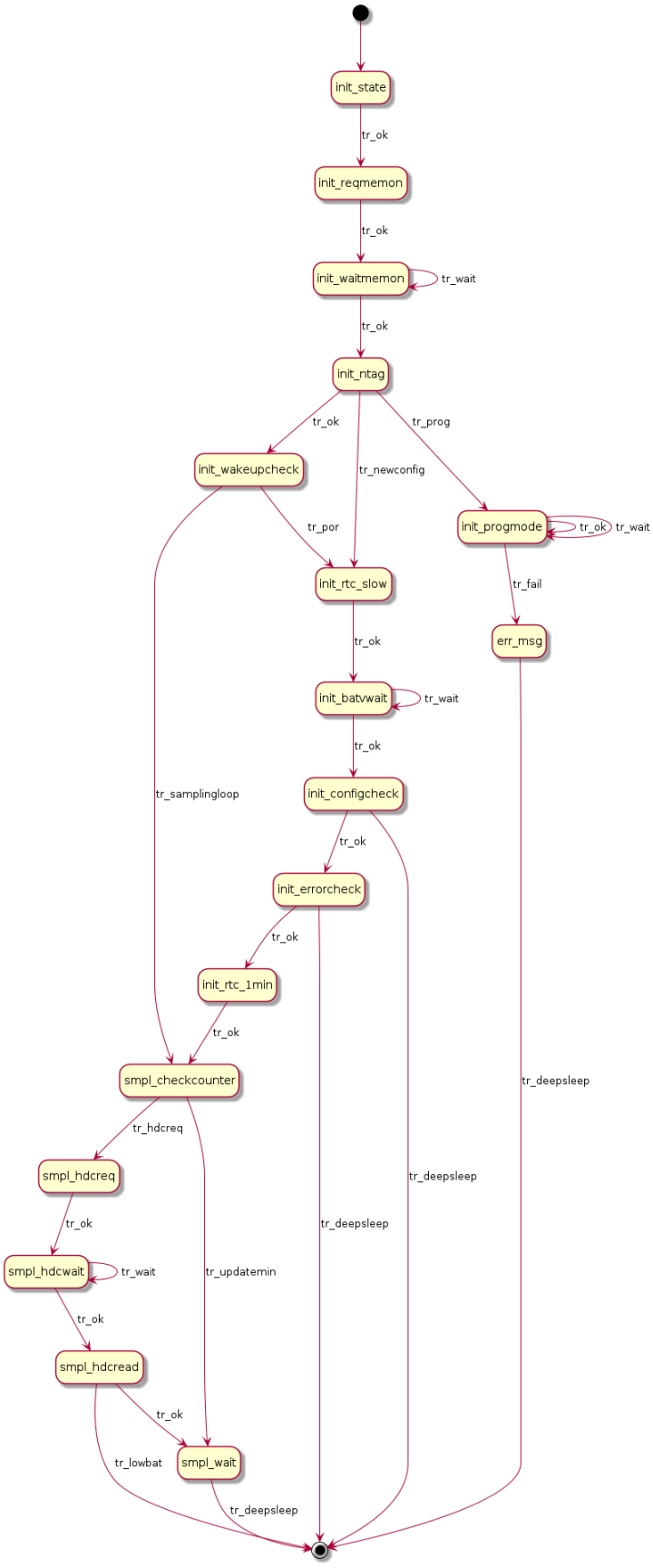


Fig. 1: The state machine in *main()*.

CHAPTER

FOUR

STARTUP

CHAPTER

FIVE

OPERATING MODES

5.1 Primary

The software progresses into the loop at the bottom of the *State Chart*:

- Programming mode is not entered because the nPRG pin is deasserted.
- Tag configuration is present according to *init_configcheck()*.
- No evidence of repeated resets has been found by *init_errorcheck()*.

In normal operation the tag is updated periodically with calls to cuplCodec. To conserve power the MSP430 Real Time Clock (RTC) is set to generate interrupts at one minute intervals. The RTC is clocked by the 32.768 kHz watch crystal. A majority of time is spent waiting in standby (LPM3) for the next RTC interrupt. The VMEM domain that includes the NT3H2211 NFC tag and the HDC2021 is powered off during this time. Therefore cuplTag draws little more than 1.43uA; equal to the MSP430 consumption in LPM3 (MSP430Datasheet).

5.1.1 Every Minute

1. cuplTag wakes up from standby (LPM3).
2. Minute counter is incremented.
3. The VMEM domain is powered on.
4. A call is made to *configNfc_check()*. This checks if an NDEF text record (assumed to contain configuration data) is present on the tag. If so, a reset occurs.
5. A call is made to `enc_setelapsed()` in cuplCodec. The minuteselapsed field (CODEC_FEAT_26) of the cuplCodec URL is updated.
6. The VMEM domain is powered off.
7. cuplTag returns to LPM3.

5.1.2 At the Sample Interval (in minutes)

1. cuplTag wakes up from standby (LPM3).
2. Minute counter is reset to 0.
3. The VMEM domain is powered on.
4. A call is made to *configNfc_check()*. This checks if an NDEF text record (assumed to contain configuration data) is present on the tag. If so, a reset occurs.
5. A sample is requested from the humidity sensor with *hdc2010_startConv()*.
6. The MSP430 waits in LPM3 until the DRDY line of this sensor is asserted.
7. The sample is read from the humidity sensor with *hdc2010_read_temp()*.
8. A call is made to `enc_pushSample()` in cuplCodec. The sample is written to the circular buffer inside the cuplCodec URL. The minuteselapsed field is reset to 0.
9. If the circular buffer has wrapped around to the start, then a call is made to `nvparams_cresetsperloop()`.
10. The VMEM domain is powered off.
11. cuplTag returns to LPM3.

Configuration file check Block 1 of the NT3H2211 is read via I2C. If it contains a text record, then it is assumed that a configuration file has been written. cuplTag resets to read the configuration file.

5.2 Secondary

The secondary operating mode is programming mode. The state *init_progMode()* is entered when the nPRG pin is low after reset. A reset is triggered at power on or by a low pulse on the nRESET pin (see *Pinout*). The only way to leave programming mode is to trigger a reset. This is done either via the aforementioned means or by sending the soft-reset command.

The serial port is active in this state and not in any other to save power. Connect with these settings:

Setting	Value
Baudrate	9600
Parity	None
Stop bit	1
Flow control	Off

A simple command and response scheme is used. Basic commands have 3 characters:

Character	Description	Note
<	Start character	
z	Command ID	Any character in the range a-z, A-Z and 0-9
>	End character	

Configuration string commands add a parameter string:

Character	Description	Note
<	Start character	
b	Command ID	Any character in [a-z, A-Z, 0-9]
:	Parameter prefix	
ABcd1234	Parameter string	Up to 64 characters in [a-z, A-Z, 0-9]
>	End character	

Responses take a similar format to commands, starting with a '<' character and ending with a '>'.

A human-readable ASCII format was chosen because very little data is transacted. It is useful to be able to send and receive commands through the terminal window without having to encode and decode packets.

5.2.1 Basic Commands

Command	Name	Response	Example	Description
<x>	Version	<HWVER_FWVER_CODECVER>	<HT04_F2_C1>	Hardware, firmware and codec versions
<y>	EnterBL	None		Enter the MSP430 UART bootloader
<z>	SoftReset	None		Reset the MSP430

5.2.2 Error Response

The cuplTag firmware responds with '<e>' if it has failed to parse a command.

5.2.3 Configuration Commands

See configuration strings.

The *State Chart* shows a theoretical transition into an error state. This can only occur if the UART state table is incomplete.

CHAPTER

SIX

ERROR CONDITIONS

Before entering *normal operation* some checks are made. If any of these fail:

1. An error message is written to the dynamic tag. This is either:
    - An NDEF text record with a description of the error.
    - The cuplCodec URL record without the circular buffer.
2. cuplTag shuts down by entering LPM4 (deep sleep).

Battery life is conserved until the user attempts to read the tag and discovers the error.

6.1 Configuration Check Failed

After cuplTag has been erased and programmed anew, the variable in non-volatile memory `allwritten` is 1. Each time a valid configuration string is received, its corresponding bit is set in the RAM-based variable `writtenfields`.

For example:

- Bit 0 is set in response to serial string.
- Bit 1 is set in response to the secret key.

When all required bits have been set in `writtenfields`, its non-volatile counterpart `allwritten` is cleared.

At startup `nvparams_allwritten()` returns 1 if `allwritten` is cleared. This means that all configuration strings have been set in order for the cuplCodec Encoder to run.

The error cannot be communicated by writing a URL to the dynamic tag. The base URL field has not been written and there is no guarantee a default URL will point to a web server.

A short NDEF text record is written instead: `Config check failed. See cuplTag documentation.`

6.2 Voltage Check Failed

cuplTag measures the voltage and will not continue if it is below a configurable threshold.

6.3 Repeated Resets caused by an Error

Flash memory on the dynamic tag must be protected from repeated writes. This may occur if a fault occurs repeatedly that causes a reset. For example:

1. A brownout reset occurs whilst the dynamic tag is been written.
2. The tag resets and start to write to the dynamic tag again. The reset reoccurs.

If unchecked this cycle can go around many times each second. This will cause the dynamic tag to have worn out before the fault can be addressed. The cuplTag employs a "last ditch" protection feature to avoid this.

6.4 Invalid State Transition

This error may be encountered by a programmer but should never be seen by an end-user!

A function should never request a state transition that is not defined in the state table.

If this does happen, a catch-all state is entered `err_reqmemon()`. The dynamic tag is powered up and an NDEF text record is written: `Invalid state transition`.

cuplTag subsequently enters its end state and powers down to LPM4.

CHAPTER

SEVEN

CONFIGURATION

7.1 Mechanisms

7.1.1 Serial

Configuration string can be transmitted as commands in *Programming Mode*.

7.1.2 NFC

A message containing one NDEF text record can be written to the tag. A number of 3rd party apps NFC writer apps are capable of this. In addition, this can be done using WebNFC using wsfrontend.

The text record is comprised of one or more configuration strings described below. The cuplTag firmware checks for the presence of this record each minute (see *Normal Operation*). This consumes negligible power because the VMEM domain is powered up for writing the NTAG anyway. It is also simpler than using the Field Detect interrupt from the NTAG.

If the text record is found, a soft reset is triggered and the text record is parsed at startup; any new parameters written to non-volatile memory before use by cuplCodec.

7.2 Configuration Strings

7.2.1 Base URL

Command ID	b
Parameter Length	up to 64
Parameter value	Any string

Example: <`b:localhost:5000`>

7.2.2 Serial

Command ID	w
Parameter Length	8
Parameter value	Any URL-safe Base64

Example: <w:KEG2lARW>

7.2.3 Sample Interval in Minutes

Command ID	t
Parameter Length	5 (max)
Parameter value	Integer (16-bit)

Example: <t:20>

7.2.4 HMAC Secret Key

Command ID	s
Parameter Length	16
Parameter value	Any URL-safe Base64

Example: <s:4EOBdBWTsjeFZTm3>

7.2.5 Use HTTPS

Command ID	h
Parameter Length	1
Parameter value	0 (HTTPS disabled), 1 (HTTPS enabled)

Example: <h:1>

7.2.6 Use HMAC

Command ID	i
Parameter Length	1
Parameter value	0 (HMAC disabled), 1 (HMAC enabled)

Example <i:1>

CHAPTER

EIGHT

REFERENCE

8.1 Main

Defines

CS_SMCLK_DESIRED_FREQUENCY_IN_KHZ
Target frequency for SMCLK in kHz.

CS_XT1_CRYSTAL_FREQUENCY
Resonant frequency of the XT1 crystal in kHz.

CS_XT1_TIMEOUT
Timeout for XT1 to stabilise at the resonant frequency in SMCLK cycles.

CP10MS
ACLK Cycles Per 10 MilliSeconds. Assumes ACLK = 32768 kHz and a divide-by-8.

EXIT_STATE
State machine exit state.

ENTRY_STATE
State machine entry state.

FRAM_WRITE_ENABLE

FRAM_WRITE_DISABLE

Typedefs

typedef enum *state_codes* **tstate**

typedef enum *ret_codes* **tretcode**

typedef enum *event_codes* **tevent**

Enums

enum **state_codes**
Values:

enumerator **sc_init**

enumerator **sc_init_reqmemon**

enumerator **sc_init_waitmemon**

enumerator **sc_init_ntag**

enumerator **sc_init_progmode**

enumerator **sc_init_configcheck**

enumerator **sc_init_errorcheck**

enumerator **sc_init_wakeupcheck**

enumerator **sc_init_batvwait**

enumerator **sc_init_rtc_slow**

enumerator **sc_init_rtc_1min**

enumerator **sc_smpl_checkcounter**

enumerator **sc_smpl_hdcreq**

enumerator **sc_smpl_hdcwait**

enumerator **sc_smpl_hdcread**

enumerator **sc_smpl_wait**

enumerator **sc_err_msg**

enumerator **sc_end**

enum **ret_codes**
Values:

enumerator **tr_ok**

enumerator **tr_prog**

enumerator **tr_newconfig**

enumerator **tr_hdcreq**

enumerator **tr_updatemin**

enumerator **tr_deepsleep**

enumerator **tr_lowbat**

enumerator **tr_fail**

enumerator **tr_samplingloop**

enumerator **tr_por**

enumerator **tr_wait**

enum **event_codes**
Values:

enumerator **evt_none**

enumerator **evt_timerfinished**

enumerator **evt_hdcint**

Functions

void **fram_write_enable**()

void **fram_write_disable**()

tretcode **init_state**(*tevent*)

tretcode **init_reqmemon**(*tevent*)

tretcode **init_waitmemon**(*tevent*)

tretcode **init_ntag**(*tevent*)

tretcode **init_progmode**(*tevent*)

tretcode **init_configcheck**(*tevent*)

tretcode **init_errorcheck**(*tevent*)

tretcode **init_wakeupcheck**(*tevent*)

tretcode **init_batvwait**(*tevent*)

tretcode **init_rtc_slow**(*tevent*)

tretcode **init_rtc_1min**(*tevent*)

tretcode **smpl_checkcounter**(*tevent*)

tretcode **smpl_hdcreq**(*tevent*)

tretcode **smpl_hdcwait**(*tevent*)

tretcode **smpl_hdcread**(*tevent*)

tretcode **smpl_wait**(*tevent*)

tretcode **err_msg**(*tevent*)

tretcode **end_state**(*tevent*)

tstate **lookup_transitions**(*tstate* curstate, *tretcode* rc)

static void **writetxt**(const char *msgptr, int len)

static void **wdog_kick**()

static void **start_timer**(unsigned int intervalCycles)

static void **memoff**()

static *tretcode* **reqmemon**(*tevent* evt)

static *tretcode* **waitmemon**(*tevent* evt)

void **main**(void)

void **TIMER1_B0_ISR**(void)

if ((P1IN &BIT1)==0)

Variables

int **timerFlag** = 0
: Flag set by the Timer Interrupt Service Routine.

int **hdcFlag** = 0
: Flag set by the HDC2021 humidity sensor data-ready Interrupt Service Routine.

int **minutecounter** = 0

Incremented each time the sampling loop is run.

const char **ndefmsg_progmode**[] = {0x03, 0x3D, 0xD1, 0x01, 0x39, 0x54, 0x02, 0x65, 0x6E, 0x50, 0x72, 0x6F, 0x67, 0x72, 0x61, 0x6D, 0x6D, 0x69, 0x6E, 0x67, 0x20, 0x4D, 0x6F, 0x64, 0x65, 0x2E, 0x20, 0x43, 0x6F, 0x6E, 0x6E, 0x65, 0x63, 0x74, 0x20, 0x74, 0x6F, 0x20, 0x73, 0x65, 0x72, 0x69, 0x61, 0x6C, 0x20, 0x70, 0x6F, 0x72, 0x74, 0x20, 0x61, 0x74, 0x20, 0x39, 0x36, 0x30, 0x30, 0x20, 0x62, 0x61, 0x75, 0x64, 0x2E, 0xFE}

const char **ndefmsg_noconfig**[] = {0x03, 0x2D, 0xD1, 0x01, 0x29, 0x54, 0x02, 0x65, 0x6E, 0x43, 0x6F, 0x6E, 0x66, 0x69, 0x67, 0x20, 0x63, 0x68, 0x65, 0x63, 0x6B, 0x20, 0x66, 0x61, 0x69, 0x6C, 0x65, 0x64, 0x2E, 0x20, 0x53, 0x65, 0x65, 0x20, 0x63, 0x75, 0x70, 0x6C, 0x54, 0x61, 0x67, 0x20, 0x64, 0x6F, 0x63, 0x73, 0x2E, 0xFE}

const char **ndefmsg_badtrns**[] = {0x03, 0x27, 0xD1, 0x01, 0x23, 0x54, 0x02, 0x65, 0x6E, 0x45, 0x72, 0x72, 0x6F, 0x72, 0x3A, 0x20, 0x49, 0x6E, 0x76, 0x61, 0x6C, 0x69, 0x64, 0x20, 0x73, 0x74, 0x61, 0x74, 0x65, 0x20, 0x74, 0x72, 0x61, 0x6E, 0x73, 0x69, 0x74, 0x69, 0x6F, 0x6E, 0x2E, 0xFE}

tretcode (***state_fcns**[])(*tevent*) = {*init_state*, *init_reqmemon*, *init_waitmemon*, *init_ntag*, *init_progmode*, *init_configcheck*, *init_errorcheck*, *init_wakeupcheck*, *init_batvwait*, *init_rtc_slow*, *init_rtc_1min*, *smpl_checkcounter*, *smpl_hdcreq*, *smpl_hdcwait*, *smpl_hdcread*, *smpl_wait*, *err_msg*, *end_state*}

struct *transition* **state_transitions**[]

struct **transition**

Public Members

tstate **src_state**

tretcode **ret_code**

tstate **dst_state**

8.2 HDC2010

Functions

int **hdc2010_init**()

uint32_t **hdc2010_read_temp**(int*, int*)

uint32_t **hdc2010_read_humidity**()

int **hdc2010_read_whoami**()

int **hdc2010_startconv**()

Defines

DEVADDR

TEMPL_REGADDR

TEMPH_REGADDR

HUML_REGADDR

HUMH_REGADDR

INT_REGADDR

TEMPMAX_REGADDR

HUMMAX_REGADDR

INTEN_REGADDR

TEMPOFFSETADJ_REGADDR

HUMOFFSETADJ_REGADDR

TEMPTHRL_REGADDR

TEMPTHRH_REGADDR

RHTHRH_REGADDR

RHTHRL_REGADDR

RSTDRDYINTCONF_REGADDR

MEASCONF_REGADDR

MANFIDL_REGADDR

MANFIDH_REGADDR

DEVIDL_REGADDR

DEVIDH_REGADDR

MEAS_TRIG_BIT

DRDY_STATUS_BIT

INTEN_DRDYEN_BIT

DRDY_SOFT_RES_BIT

DRDY_ODR_NOREPEAT_BITS

DRDY_ODR_5HZ_BITS

DRDY_HEATER_BIT

DRDY_INTEN_BIT

DRDY_INTPOL_BIT

Functions

int **hdc2010_startconv**()

int **hdc2010_init**()

uint32_t **hdc2010_read_temp**(int *tempdeg, int *rh)

uint32_t **hdc2010_read_humidity**()

int **hdc2010_read_whoami**()

8.3 I2C

Functions

void **i2c_init**()

void **i2c_off**()

int **i2c_readreg**(uint8_t slaveAddr, uint8_t mema, uint8_t rega)

int **i2c_read_block**(uint8_t slaveAddr, uint8_t regOffset, uint8_t bytesToRead, uint8_t *rxData, uint8_t rega)

int **i2c_write_block**(uint8_t slaveAddr, uint8_t regOffset, uint8_t bytesToWrite, uint8_t *txData)

uint8_t **i2c_read8**(uint8_t slaveAddr, uint8_t regOffset)

uint16_t **i2c_read16**(uint8_t slaveAddr, uint8_t regOffset)

uint16_t **i2c_read32**(uint8_t slaveAddr, uint8_t regOffset, uint16_t *temp, uint16_t *hum)

uint8_t **i2c_write8**(uint8_t slaveAddr, uint8_t regOffset, uint8_t writeData)

Defines

EUSCI_BASE

Functions

void **i2c_init**()

void **i2c_off**()

uint8_t **i2c_write8**(uint8_t slaveAddr, uint8_t regOffset, uint8_t writeData)

int **i2c_readreg**(uint8_t slaveAddr, uint8_t mema, uint8_t rega)

void **i2c_send_start**(uint8_t slaveAddr)

int **i2c_write_block**(uint8_t slaveAddr, uint8_t regOffset, uint8_t bytesToWrite, uint8_t *txData)

int **i2c_read_block**(uint8_t slaveAddr, uint8_t regOffset, uint8_t bytesToRead, uint8_t *rxData, uint8_t rega)

uint8_t **i2c_read8**(uint8_t slaveAddr, uint8_t regOffset)

uint16_t **i2c_read16**(uint8_t slaveAddr, uint8_t regOffset)

uint16_t **i2c_read32**(uint8_t slaveAddr, uint8_t regOffset, uint16_t *temp, uint16_t *hum)

void **USCIB0_ISR**(void)

Variables

uint8_t **buffer**[16] = {0}

uint8_t **bytesLength** = 0

uint8_t **gbl_regOffset** = 0

bool **restartTx** = false

bool **nackFlag** = false

bool **stopFlag** = false

bool **restartRx** = false

8.4 Stat

Functions

void **stat_rdrstcause**()

int **stat_rstcause_is_lpm5wu**()

unsigned int **stat_get**(bool *err, bool *borsvs, int resetsalltime)

void **stat_setscantimeout**()

void **stat_setclockfailure**()

Functions

void **stat_rdrstcause**()

int **stat_rstcause_is_lpm5wu**()

void **stat_setclockfailure**()

void **stat_setscantimeout**()

unsigned int **stat_get**(bool *err, bool *borsvs, int resetsalltime)

Variables

unsigned int **rstcause** = 0

8.5 Config NFC

Functions

int **configNfc_check**()

int **configNfc_readText**()

Defines

PAYLOADSTART_SHORTREC_INDEX

RECORDTYPE_SHORTREC_INDEX

PAYLOADLEN_SHORTREC_INDEX

Enums

enum **searchState_t**
Values:

enumerator **startCharSearch**

enumerator **cmdFound**

enumerator **dataFound**

enumerator **endCharSearch**

Functions

int **confignfc_check**()

int **confignfc_readtext**()

Variables

static char **readbuffer**[BLKSIZE]

unsigned char **msgblock**[64]

int **readfromtag** = 0

8.6 Comms UART

Warning: doxygenfile: Cannot find file "comms/comms_uart.h

Warning: doxygenfile: Cannot find file "comms/comms_uart.c

8.7 Non-Volatile Parameters

Warning: doxygenfile: Cannot find file "comms/nvparams.h

Warning: doxygenfile: Cannot find file "comms/nvparams.c

CHAPTER

NINE

HT05 MODULE

9.1 Pinout

Pin	Name	Direction	Description	Note
1	TX	Output	UART transmit	
2	RX	Input	UART receive	Also ADC input A6
3	nPRG	Input	Select programming mode	Active low
4	TCK	Input	Spy-Bi-Wire Clock	Also TEST signal for BSL entry
5	TDIO	I/O	Spy-Bi-Wire Data	Also nRESET signal
6	GND		GND	
7	VDD	Input	Power in	
8+	NC		Not connected	

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

L

M

N

P

R

S

T

U

W