I^2C Implementation Using the N2HET

Haixiao Weng MCU Safety Application

ABSTRACT

This application report describes how the Hercules N2HET peripheral can be used to implement I^2C master mode with interrupt capability and zero CPU overhead. The solution applies to all the Hercules MCUs that have N2HET modules such as TMS570 series and RM series devices.

Contents

Figures

Tables

1 Introduction

The N2HET peripheral is a complex high-performance RISC coprocessor that operates independently from the main Cortex^{TM} R4F CPU and can be used to implement complex timed I/O operations running in the background. This is the enhanced version of the HET in TMS470(M) series and the NHET in TMS570LS20x series. More information on the N2HET can be found in the Hercules TRM and datasheet.

In this application report, the N2HET is used to implement I^{2}C master interface. The goal is to provide hardware I^{2}C-module-like functionality with independent background transmission/reception, features include:

- Speed: support both I^{2}C standard and fast-mode.
- Interrupt: receive, transmit, NACK, and timeout interrupt.
- Address mode: 7-bit address space.
- Clock stretching
- Types of messages:
  - Single message where a master writes data to a slave;
  - Single message where a master reads data from a slave;
  - Combined messages, where a master issues at least two reads and/or writes to slaves repeated START bit mode.

The N2HET program runs in the background, independent of the main Cortex^{TM} R4F CPU, and performs all the tasks associated with the I^{2}C communication. The incoming data stream delivered to an N2HET device I/O pin is decoded. Upon the reception of a full character, the Cortex^{TM} R4F CPU program is notified, and it then can directly fetch the entire received data byte from the N2HET internal RAM. For outgoing data streams, the Cortex^{TM} R4F CPU just passes the data to be transmitted to the N2HET program and initiates the transmission. After this, the entire transmission is handled by the N2HET program, and a data stream is output to an N2HET device I/O pin. During this process, the Cortex^{TM} R4F CPU is free for application-related tasks. Figure 1 shows a conceptual overview of the interaction among different components. It illustrates the hardware setup for the attached example as well.

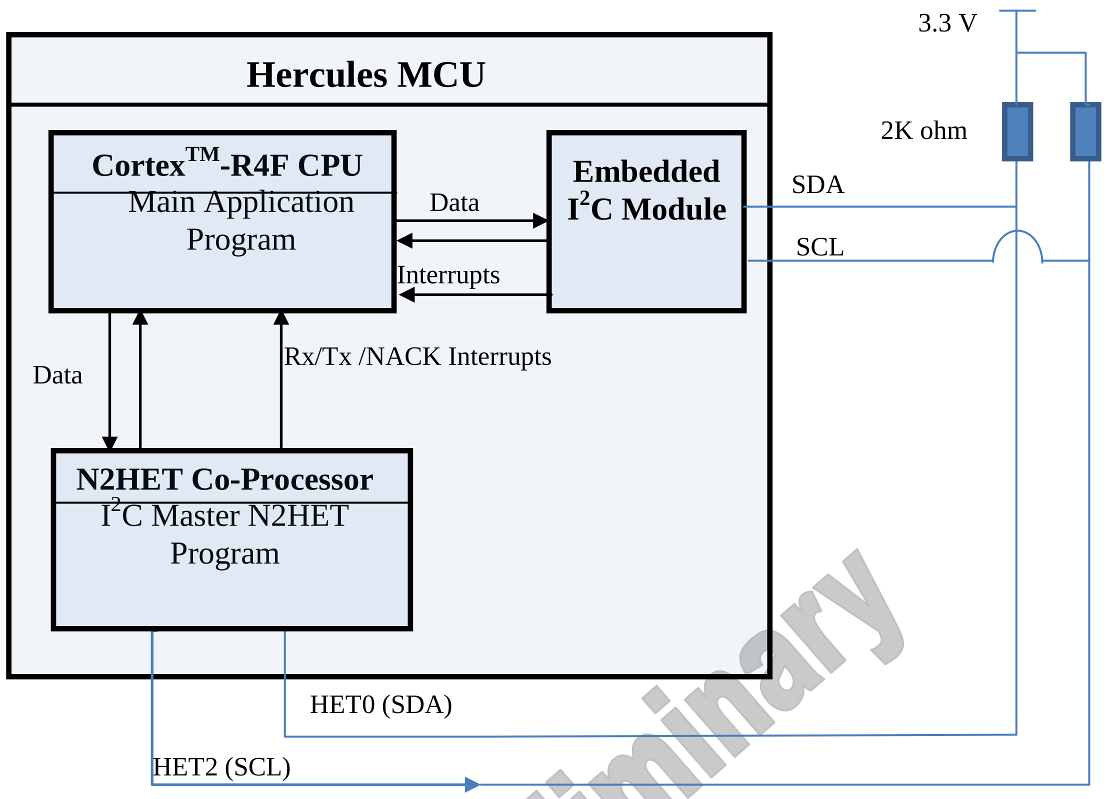


Figure 1. Hercules N2HET Emulated I^{2}C Implementation Overview

2 Examples

Two example CCS projects are provided to demo the N2HET emulated I^{2}C master, one works for the TMS570LS series and the other one works for the RM4x series. The N2HET clock is set to 89.6MHz and the N2HET emulated I^{2}C speed is set to fast mode – 400kbit/s. Suppose you follow the hardware setup in Figure 1, import the examples to CCS5.3, download the code to a TMS570LS1227, TMS570LS3137, RM46L852 or RM48L950 device, and run it, the following waveform will show up on the SDA and SCL connections. The pink curve represents the SDA line and the blue curve represents the SCL line. The waveform includes three parts (please check the attached excel sheet for details):

1) the N2HET emulated I^{2}C write 3 bytes to the slave
2) the N2HET emulated I^{2}C read 3 bytes from the slave
3) a combined message, the N2HET emulated I^{2}C write 2 bytes to the slave, following with a repeated start, then, the N2HET emulated I^{2}C read 2 bytes from the slave.

If you port the examples to a different Hercules device other than those listed in the form paragraph, please check the device datasheet to make sure that your device's HCLK frequency can tolerance 179.2MHz. Otherwise, please adjust the PLL settings to slow down the HCLK, meanwhile, adjust the HCLK to VCLK2 ratio in CLKCNTL register (@0xFFFFFFD0) and HETPFR register (@0xFFF7B804 for N2HET1 or @0xFFF7B904 for N2HET2) to keep the same LRP – 0.625ns. If the use case requires a slower I^{2}C speed (e.g. 100kbit/s), please slow down the LRP 4 times as well.

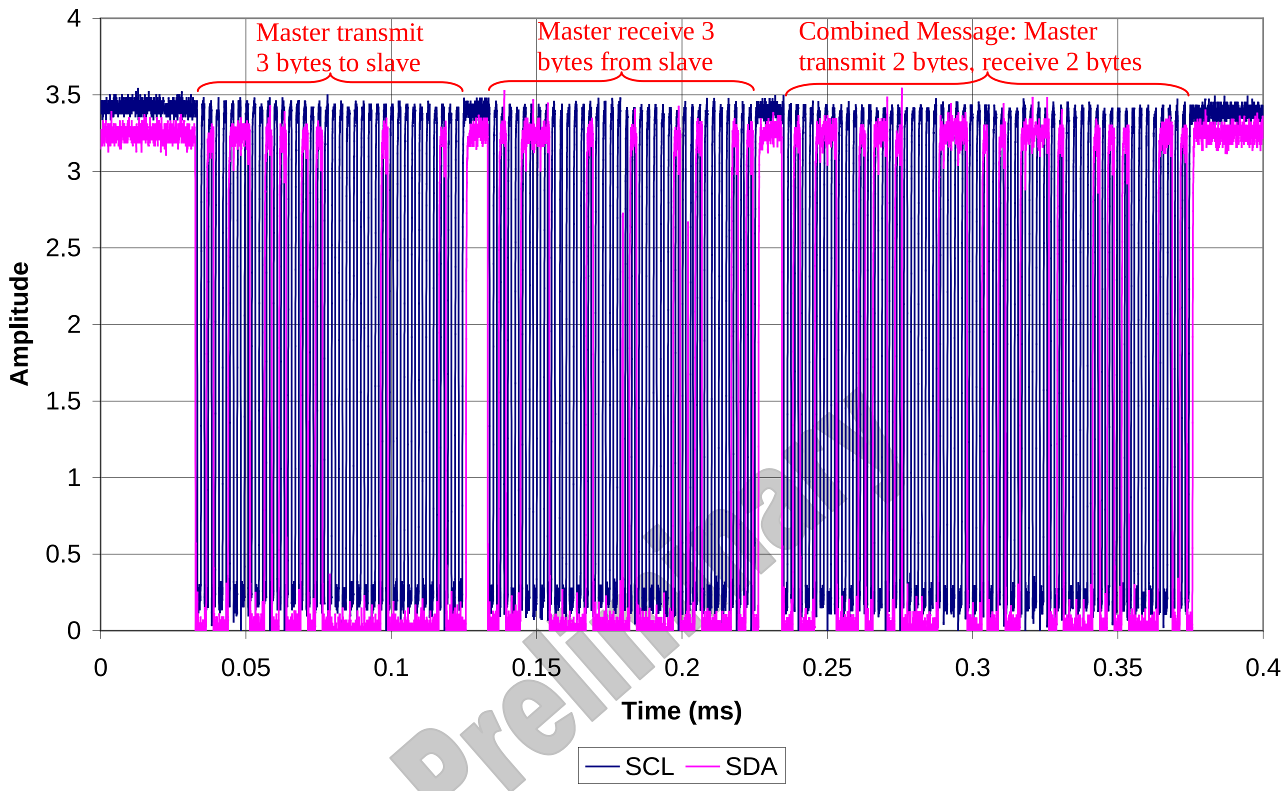


Figure 2. The N2HET Emulated I^{2}C Example Waveform

3 CPU Side Software Description

This section discusses Cortex^{TM} R4F CPU side software of the attached example, and does not explain the N2HET program. The N2HET program will be explained in next section.

I. Initialize the I^{2}C module.

The attached example uses the Hercules MCU I^{2}C module as the I^{2}C slave. So, what the application code does first is to initialize the on-chip I^{2}C module, setting the PINMUX and configuring the I^{2}C. In sys_main.c:

```
    (*(volatile unsigned int *)(0xFFFFEA38))=0x83E70B13;
    (*(volatile unsigned int *)(0xFFFFEA3C))=0x95A4F1E0;

    *(volatile unsigned int *) 0xFFFFEB10 = 0x02020101; // SCL 0[17], SDA 0[25],

    (*(volatile unsigned int *)(0xFFFFEA38))=0x0;
    (*(volatile unsigned int *)(0xFFFFEA3C))=0x0;

    i2cInit(0,    I2C2_ADDR);/** - set i2c mode */
```

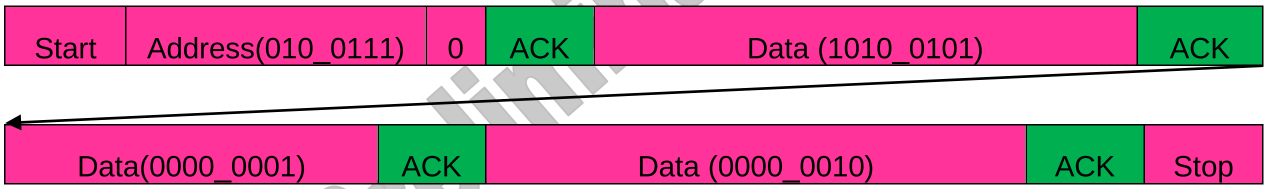
II. Initialize the N2HET

hetInit() sets N2HET pin 0 and 2 as open drain output, load the N2HET program into N2HET RAM. Then, the application enables the CPU IRQ and N2HET interrupt and start the N2HET Program. In sys_main.c:

```
    hetInit();
    hetREG1->INTENAS = 0xFFFFFFFFU;//enable the HET interrupt
    asm(" cpsie i"); //enable the CPU IRQ interrupt
    hetREG1->GCR = 0x01010001;//start HET program
```

III. Start test for “single message where a master writes data to a slave”

Table 1.   I2C master write three data to slave



To start a transfer, the Cortex^{TM} R4F CPU needs to write Start bit, address bits, R/W bit, numer of bytes to be transferred, and Stop bit to the transmit buffer of N2HET emulated I^{2}C Master. The *HetI2CPutAddr()* organize all of them into one write to the N2HET RAM.

HetI2CPutAddr() has a few input parameters, in the following example: I2C2_ADDR is the slave I^{2}C 7-bit address; RW represents whether it is master transmit message (‘0’) or it is master receive message (‘1’); NumofBytes represents the number of data to be transmitted or received; IntEna represents whether a transmit interrupt will be generated once the data is moved to shift register; StopBit represents whether the stop bit will append to the end of transfer. In this example, the parameters passed are: (0x27, 0, 3, 1, 1), which means: the slave address to be talked is 0x27, it is a master transmit action, the master will transmit 3 data to slave (the address byte itself is not included), the transmit interrupt will be generated, and the stop bit will append to the end of the transfer. In sys_main.c:

```
    HetI2CPutAddr(I2C2_ADDR, RW, NumOfBytes, IntEna, StopBit);
```

IV. Transmit 3 bytes data from N2HET emulated I^{2}C Master to the slave.

Once the address bits move to the shift register, a transmit interrupt will be generated. Inside the interrupt service routine, *HetI2CPutData()* write the data to be transmitted into the transmit buffer. In het.c:

```
void het1HighLevelInterrupt(void)
{
    uint32_t vect = hetREG1->OFF1;
    switch (vect)
    {
    case 11: /*--------------------------->      Transmit interrupt */
    if(Data_Send_HET<3)
    {
                  HetI2CPutData(*I2C1_txptr++, IntEna);
                  Data_Send_HET++;
    }
```

V. Slave receive 3 bytes data

Once the I^{2}C module receives data, an I^{2}C receive interrupt will be triggered. The received data will be copied to the main memory. In i2c.c:

```
void i2cInterrupt(void)
{
    register unsigned char vect = (i2cREG1->IVR & 0x00000007);

    /*-----------------------------------------------------------------------*/
    switch (vect)
    {
    case 0:
           i2cREG1->STR = 0x000007FF;
           break;
    case 4: /*--------------------------->      Receive interrupt */
           *I2C2_rxptr++ = i2cREG1->DRR;  // read the data
           Data_Rece_I2C++;
           break;
```

VI. Process stop bit

After all three data has been sent to the slave, the stop bit will be generated. Once the slave receives this stop bit, it will set the global variable Stop_Rece_I2C to 1. At this point, the test for “single message where a master writes data to a slave” completes. In i2c.c:

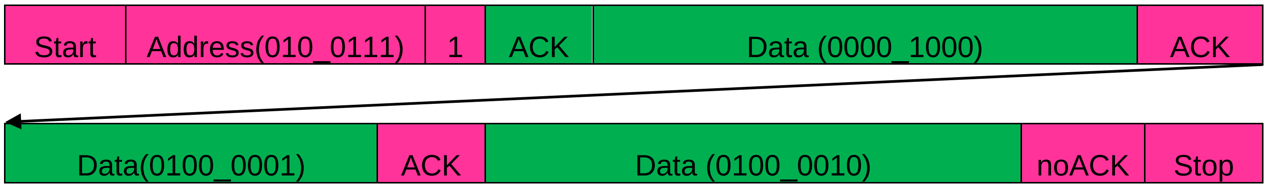
```
    case 6:  /*--------------------------->      stop condition detection interrupt
    */
           Stop_Rece_I2C = 1;
           break;
```

The application will wait for a short period and continue with the test for “single message where a slave writes data to a master”. In sys_main.c:

```
    while(Stop_Rece_I2C == 0);//wait until master transmit completes.
    Stop_Rece_I2C = 0;
    for(wait_counter=0;wait_counter<0x18;wait_counter++); //wait some time.
```

VII. Start test for “single message where a slave writes data to a master”

Table 2. I2C master read three data from slave



The message starts with a slave address write. In this case, the parameters passed are: (0x27, 1, 3, 0, 1), which means: the slave address to be talked is 0x27, it is a master receive action, the master will receive 3 data from slave (the address byte itself is not included), the transmit interrupt will not be generated, and the stop bit will append to the end of the transfer. In sys_main.c:

```
    RW = 1; //Read
    IntEna = 0;//no transmit interrupt
    HetI2CPutAddr(I2C2_ADDR, RW, NumOfBytes, IntEna, StopBit);
```

VIII. Slave sends 3 bytes data

Once the I^{2}C module receives 7-bit addressing bits and R/W bit ('1' in this case), an I^{2}C slave transmit interrupt will be triggered. The data to be transmitted will be copied to the I^{2}C transmit buffer. In i2c.c:

```
    case 5:  /*--------------------------->      Transmit interrupt */
        i2cREG1->DXR = *I2C2_txptr++;  // send the data
        Data_Send_I2C++;
        break;
```

IX. N2HET emulated I^{2}C Master receives 3 bytes data from slave.

Once the N2HET emulated I^{2}C receives the data, a receive interrupt will be generated. Inside the interrupt, the received data will be copied to the main memory. In het.c:

```
    case 15: /*--------------------------->      Receive interrupt */
        *I2C1_rxptr++ = (hetRAM1->Instruction[0x2C].Data>>8) & 0xFF;//read data
        Data_Rece_HET++;
        break;
```

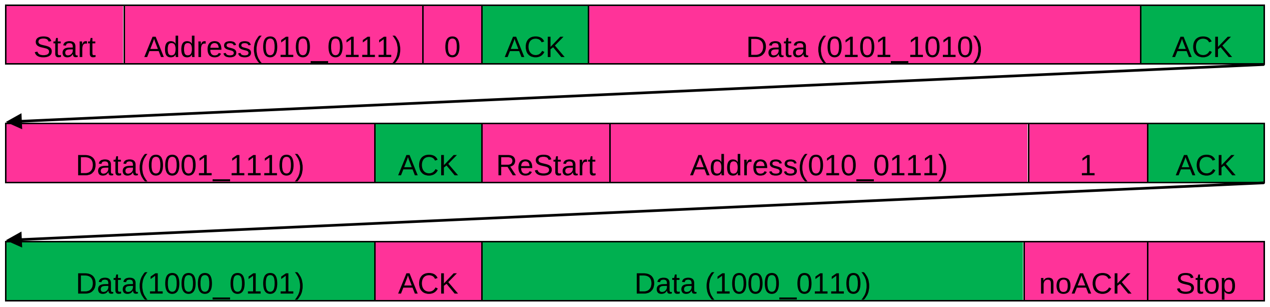
X. Process with stop bit

Same as step VI, after the transfer completes, the stop bit will be sent out. Once the slave receives the stop bit, an interrupt will be triggered. Then the application can continue with the next test - Combined messages, where a master issues at least two reads and/or writes to slaves - repeated START bit mode.

XI. Start "combined message in repeated START bit mode"

The N2HET emulated I^{2}C Master will first write 2 data bytes to slave and then read 2 data bytes from slave. Two start bit and only one stop bit will be sent out in this procedure.

Table 3. I2C master write two data to slave, re-start, read two data from slave



The message starts with a slave address write. In this case, the parameters passed are: (0x27, 0, 2, 1, 0), which means: the slave address to be talked is 0x27, it is a master transmit action, the master will transmit 2 data to slave (the address byte itself is not included), the transmit interrupt will be generated, and no stop bit append to the end of the transfer. In sys_main.c:

```
RW = 0; //write
IntEna = 1;//transmit interrupt
StopBit = 0; //no stop bit for the first transfer
NumOfBytes = 2; //write 2 data, 1 repeat address, read 2 data
HetI2CPutAddr(I2C2_ADDR, RW, NumOfBytes, IntEna, StopBit);
```

Please note that in repeated START bit mode, only the final start bit has matching stop bit.

XII. N2HET emulated I^2C Master transmit 2 bytes data to the slave. After that, it receives 2 bytes data from the slave.

This step is similar to step VIII and IX. After two bytes of data is sent out to the slave, the application initiates another transfer – receive two bytes of data from slave. Please note that, no stop bit is inserted between the write and read. After all the transfer complete, the application code reaches a while(1) loop. The user can check the received data I2C1_RxData and I2C2_RxData against the data transmitted I2C2_TxData and I2C1_TxData.

4 N2HET Emulated I^2C

This section explains the N2HET program. You can skip it if you use the example as it is.

This N2HET program emulates the I^2C master as a state machine. Table 4 illustrates the key part of a typical I^2C master and the related N2HET state machine index. Table 5 explains the pseudo C functions used in Table 4.

Table 4. N2HET Emulated I^2C State Machine

I^2C Master Function	Pseudo C Expression	N2HET State Machine Index

Generate START Bit	while (Data_Valid() == 0);
	Writ_SDA(1);
		State0

	Writ_SCL(1);	State1

	`while (Read_SCL()==0) {`
	`// Clock stretching`
	`// Consider timeout here`
		State2

	`}`
	`Writ_SDA(0);`	

	`Writ_SCL(0);`	State3

Write/Read a data/address bit	`Writ_SDA(x);`
	`//read: x=1, write: x=bit to be written`
		State4

	`Writ_SCL(1);`
	`while (Read_SCL()==0) {`
	`// Clock stretching`
	`// Consider timeout here`
	`}`
	`Read_SDA();//Read data bit in`
		State5

	`Writ_SCL(0);`	State6
Generate STOP Bit	`Writ_SDA(0);`	State7

	`Writ_SCL(1);`
	`while (Read_SCL()==0) {`
	`// Clock stretching`
	`// Consider timeout here`
	`}`
		State8

	`Writ_SDA(1);`	State9

Table 5. Pseudo C Expression Description

Pseudo C Expression	Description
Data_Valid()	Check whether the transmit buffer has valid data not
Writ_SDA(x);	Write a bit ‘x’ to Pin SDA. Write ‘1’ doesn’t impact the Open-Drain Pin Status. Write ‘0’ clear SDA.
Writ_SCL(x);	Write a bit ‘x’ to Pin SCL. Write ‘1’ doesn’t impact the Open-Drain Pin Status. Write ‘0’ clear the SCL.
Read_SCL();	Read the current status of Pin SCL.
Read_SDA();	Read the current status of Pin SDA.

Figure 3 discuss the N2HET program flow of the I^{2}C master implementation, in other words, how the N2HET program switches among those states in Table 4. Note that when it is indicated that data is stored at or loaded from an N2HET program location, this always refers to the data field that is embedded in the instruction.

N2HET program State0, which indicates the emulated I^{2}C is in idle mode, constantly polls the transmit buffer (data field of Master_Start). If the “Valid” bit is ‘1’ in this buffer, it will process and copy the transmit buffer into the data shift register (data field of SDA_SHFT). A transmit interrupt can be issued to the Cortex^{TM} R4F CPU at this point. Meanwhile, the clock shift register (data field of SCL_SHFT) is initialized as well. If the data to be transmitted is an address, it will save the “R/W” bit and “Stop” bit in data field of State0 and the “Num_of_Bytes” to be transferred in the data field of ByteRecTran. For more details, please refer to the 5 transmit/receive buffer description. The N2HET program will switch to State1 to send the start bit or switch to State4 to transmit/receive data according to the start bit settings in the transmit buffer.

The transmitted data bit is sent to SDA pin after the falling edge of the SCL and the received data bit is sampled (shift in) after the rising edge of the SCL. As described by the clock stretching, the slave may hold the SCL low if it is not ready. The N2HET program implements a counter to deal with such saturation. If the counter overflows, a timeout interrupt will be issued to the Cortex^{TM} R4F CPU. Upon reception of all 8 data bits (one byte), a receive interrupt can be issued to the Cortex^{TM} R4F CPU. Then the R4F CPU can read out the data from CleanRecDat. Or, if it is a transmit message, the master expect the 9th bit to be ACK, if no ACK is received, a NACK interrupt can be generated. This N2HET program also implements a double-buffering scheme, effectively allowing the N2HET program to transmit/receive a byte while one is waiting to be written/read out by the application program.

Once all the planned data bytes are received or transmitted, the N2HET program will check for the “Stop” bit, and determine whether to shift out the stop bit. After sending out the stop bit, the I^{2}C message is over and the N2HET program will be back to the idle state waiting for command from Cortex^{TM} R4 CPU.

The I^{2}C N2HET program function takes 72 words of N2HET program memory. A minimum of 6 and a maximum of 33 (worst case) HET instructions are executed in one LRP. In the attached examples, HCLK=179.2MHz, VCLK2=89.6MHz, and the LRP is set to 0.625us to run the I^{2}C in fast mode – 400kbit/s. Therefore, each LRP only has 56 time slots. Here, we recommend only one I^{2}C fast mode master instance per N2HET. If running I^{2}C in standard mode – 100kbit/s, each LRP should have enough time slots to run 2 instances I^{2}C master.

Table 6. N2HET Emulated I^{2}C Master Summary

Feature	**Performance**	**Number of Instances**
I^{2}C Master, with clock stretching, no arbitration; Support transmit, receive and combined messages; Provide transmit, receive, NACK and timeout interrupt	400kbit/s	1 per N2HET
	100kbit/s	2 per N2HET

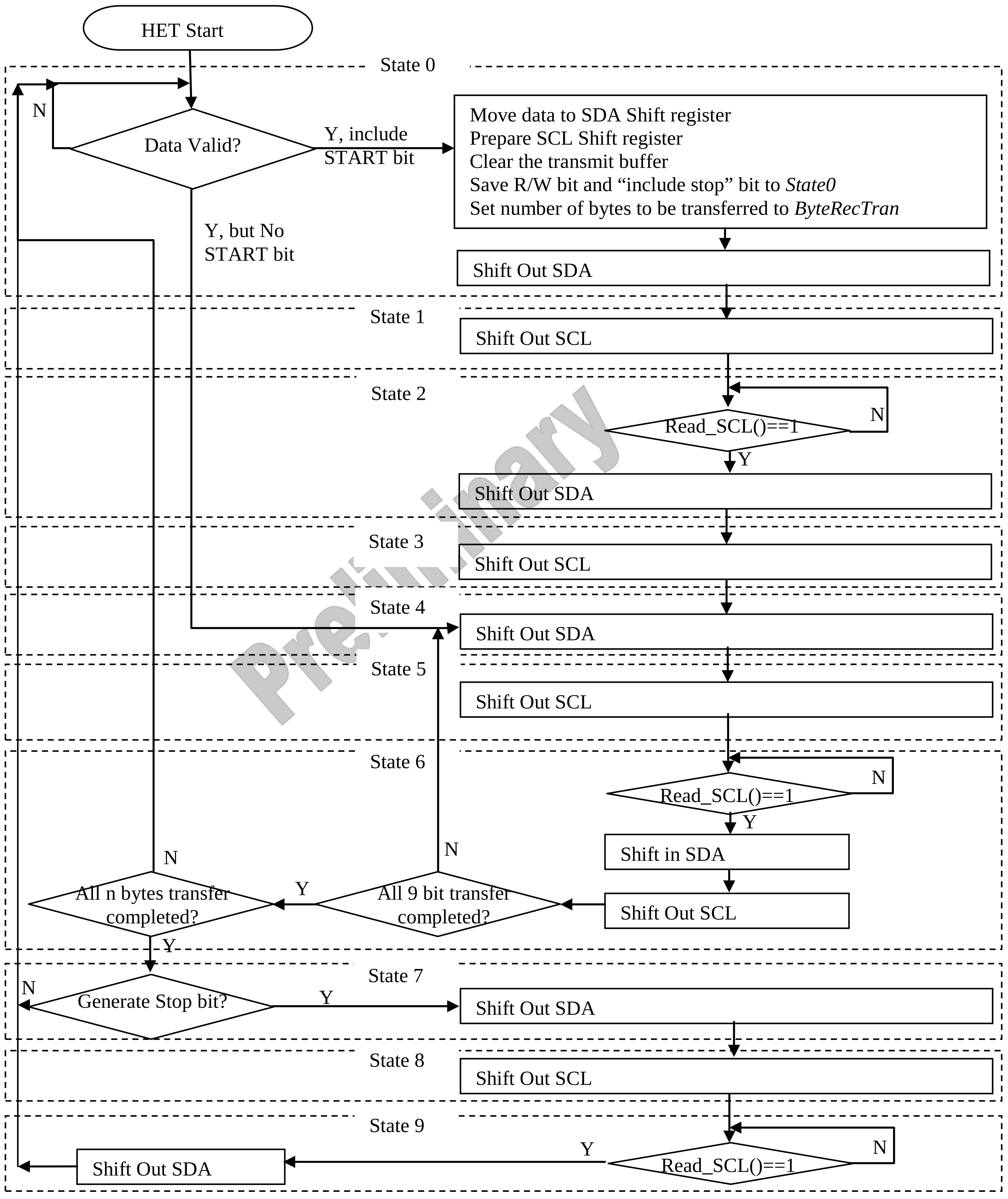


Figure 3. N2HET Emulated I^{2}C FlowChart

5 Transmit/Receive Buffer Description

I. Transmit buffer (hetRAM1->Instruction[0].Data)

31 30	29 22	21	20 19	18	17	16
Start_bit	Data_Transmit	ACK	Stop_bit	Tran_Int	Valid	Start

15	14 7	6 0
Stop	Num_of_Bytes	Reserved

Note: This buffer is write only.

Table 7. Transmit buffer Field Descriptions

Bits	Name	Value	Description
6-0	Reserved		
14-7	Num_of_Bytes		Number of bytes to be transmitted or received (exclude the address)
		0	0 byte will be transmitted or received.
		1	1 byte will be transmitted or received.
		7Fh	127 bytes will be transmitted or received.
15	Stop		Stop bit included? Only take effect when Start bit is ‘1’
		0	Don’t send out stop bit
		1	Send out stop bit
16	Start		Start bit included?
		0	Send out start bit before the data frame, which mean it is an address frame
		1	Do not send out start bit before the data frame, which mean it is not an address frame
17	Valid		Data is valid?
		0	Data is not valid. The N2HET program ignores it.
		1	Data is valid. The N2HET program move it to the shift register.
18	Tran_Int		Generate transmit interrupt?
		0	No transmit interrupt will be generated.
		1	A transmit interrupt will be generated after the data is moved to the shift register
20-19	Stop_bit	1h	Represents the stop bit (01b) to be sent out to SDA
21	ACK		Send out ACK bit?
		0	Send out ACK bit ‘0’
		1	Do not Send out ACK bit
29-22	Data_Transmit		Data to be transmitted. If Start is ‘1’, it is the 7 address bits + the R/W bit.
31-30	Start_bit	2h	Represents the start bit (10b) to be sent out to SDA

II. Receive buffer (hetRAM1->Instruction[0x2C].Data)

31–16
Reserved

15–8	7	6–0
Data_Received	ACK	Reserved

Note: This buffer is read only

Table 8. Receive buffer Field Descriptions

Bits	Name	Value	Description
6-0	Reserved		
7	ACK		Send out ACK bit?
		0	Send out ACK bit '0'
		1	Do not Send out ACK bit
15-8	Data_Received		Data byte received.
31-16	Reserved		

Reference

1. *TMS570LS1227 Technical Reference Manual* (SPNU515)
2. *TMS570LS1227 Data Sheet* (SPNS192)
3. *RM46L852 Technical Reference Manual* (SPNU514)
4. *RM46L852 Data Sheet* (SPNS185)